MASTER EN SCIENCES INFORMATIQUES

Étude d'un système de gestion de fichiers à structure arborescente

Benedetti, Massimo

Award date:
1982

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX (NAMUR)

INSTITUT D'INFORMATIQUE

ETUDE D'UN SYSTEME DE GESTION DE FICHIERS A STRUCTURE ARBORESCENTE.

Mémoire présenté par

Massimo Benedetti

en vue de l'obtention
du titre de
Licencié et Maître en Informatique.

Année académique 1981-1982

Remerciements

Je tiens à exprimer ma plus profonde gratitude à Monsieur Claude Cherton, sans qui ce travail n'aurait pu être réalisé. Sa patience, ses conseils judicieux et son aide constante tant sur le plan scientifique qu'humain m'ont permis de mener ce travail jusqu'au bout.

Ma reconnaissance s'adresse également à Messieurs Albert Orban et Joseph Primosig qui ont accepté de dactylographier une partie de ce mémoire.

Je tiens aussi à témoigner ma gratitude à Monsieur Joseph Pozza pour l'assistance apportée dans l'édition de ce mémoire.

Enfin, je ne remercierai jamais assez mon épouse pour sa patience et les efforts qu'elle a dû fournir tout au long de cette année académique.

TABLE DES MATIERES

TABLE DES MATIERES

INTRODUCTION

I N T R O D U C T I O N.

1. Le but du travail.

Le problème posé à l'origine de ce mémoire était de fournir un outil simple et efficace d'aide à la documentation des applications informatiques en général, des programmes en particulier. Ce problème nous semble très important : malgré la nécessité de disposer d'une documentation complète et tenue à jour d'uneapplication informatique, on sait que beaucoup de programmeurs laissent leurs produits mal documentés. Quand on connaît le temps que prend la rédaction de la documentation et les conditions de travail dans l'industrie (où l'on est toujours poussé à produire plus et plus vite), peut-être ne faut-il pas se contenter d'accuser les programmeurs : s'il est indispensable de leur imposer la discipline de documenter leurs prommes, cette discipline serait sans doute beaucoup mieux acceptée si la tâche leur était facilitée.

Tous les compilateurs offrent la possibilité d'inclure des commentaires dans un programme. Mais ces commentaires sont simplement retranscrits sur le listing : aucune facilité de gestion automatique de ces commentaires n'est offerte. Or,on constate qu'une application bien documentée nécessite une redondance importante de ces commentaires : par exemple, si une procédure est utilisée dans différents modules, il est utile d'en faire figurer les spécifications dans la définition de la procédure elle-même ainsi que dans chacun des modules qui l' utilise. Dans l'état actuel des choses, le seul outil général offert pour faciliter la copie de ces commentaires est l'éditeur qui ne propose, dans la grande majorité des cas, aucune facilité spécifique à ce genre de manipulation.

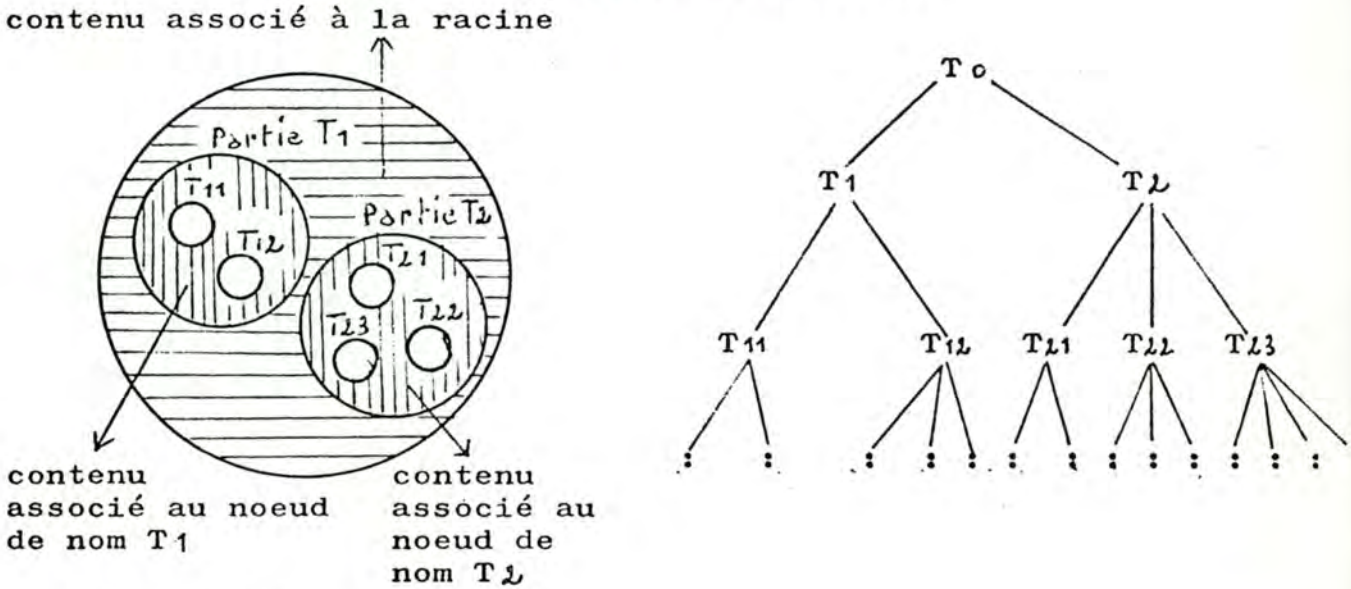
Partant de cette idée, on constate qu'elle peut s'étendre à la manipulation de textes ne constituant pas nécessairement des commentaires (par exemple des suites de déclarations qui sont souvent identiques d'un module à l'autre) ou qui ne font pas partie d'un programme (par exemple les caractéristiques d'un fichier). Dans certains cas, des facilités plus ou moins analogues à celles que nous proposons sont offertes par les compilateurs (option copy, en COBOL par exemple).

2. Les principes de l'outil proposé.

Une première idée qui ressort de ce qui précède est que l'utilisateur devrait pouvoir désigner un texte par un nom et disposer de certaines fonctions lui permettant de créer, modifier et supprimer de tels textes ainsi que d'insérer un texte désigné par son nom dans un autre. Cependant, il nous a semblé nettement insuffisant de considérer tous ces textes comme n' ayant pas de rapports les uns avec les autres. Une telle attitude conduirait à un foisonnement de noms qui devraient être tous distincts. Elle nous semble de plus tout-à-fait anti-naturelle : il est clair par exemple, que le texte constituant les spécifications fonctionnelles d'une procédure fait partie du texte de la dite procédure, et qu'il serait donc artificiel de lui donner un nom unique et de le considérer comme une entité isolée.

Il serait de loin préférable de le désigner comme une partie du texte de la procédure, partie qui elle-même serait désignée par un nom rappelant son rôle dans le texte complet.
Un texte pourra donc se décomposer en un ensemble de parties dont certaines seront utilisées plusieurs fois tandis que d' autres ne le seront qu'occasionnellement. Chacune de ces partie sera désignée par un nom et pourra elle aussi être décomposée en sous-parties. Pour refléter cette approche nous avons pensé qu'une structure arborescente conduirait au meilleur rapport complexité-efficacité. Dans cette optique, un texte sera assimilé à une structure arborescente dont les noeuds correspondron aux différentes parties du texte, seront désignés par un nom et posséderont un contenu propre.
Le schéma suivant illustre cette relation :

texte ⟷ structure arborescente



Remarques

-l'ordre des fils d'un noeud est significatif puisque chaque texte est toujours structuré en parties suivant un ordre bien spécifique.

-Il existe une contrainte sur les noms de noeuds car à partir de la racine on doit pouvoir désigner un seul noeud par la suite des noms de noeuds du chemin qui y conduit.

3. Conclusion

L'objectif de ce travail consiste donc à analyser un éditeur capable de travaux sur des fichiers à structure arborescente plutôt que sur des fichiers séquentiels.

Ce mémoire comporte deux parties :
-analyse fonctionnelle du problème
-analyse organique du problème

Dans la première partie, nous allons d'abord dégager les résultats qu'on désire obtenir et citer les différents problèmes qu' on doit résoudre pour obtenir ces résultats. On s'attachera ensuite à l'analyse fonctionnelle de ces différents problèmes et pour chacun d'entre eux nous ferons apparaître les solutions. Nous terminerons cette première partie par un résumé de la syntaxe des différentes fonctions de traitement.

Dans la deuxième partie, nous allons définir les organes du système à mettre en place : forme concrète des fichiers et des programmes. Nous commencerons par la description des structures logiques de données et nous décrirons ensuite les algorithmes des différentes fonctions de traitement en utilisant un langage proche du Pascal.

PREMIERE PARTIE : ANALYSE FONCTIONNELLE

Première partie : Analyse fonctionnelle

1. Quels sont les résultats qu'on désire obtenir ? Quels sont les problèmes qu'on doit résoudre pour obtenir ces résultats ?

1.1 Quels sont les résultats qu'on désire obtenir ?

Notre problème consiste à analyser un éditeur capable de gérer des fichiers à structure arborescente (création , modification, suppression) ; gestion comprenant la possibilité d'inclure dans un fichier tout ou partie d'un fichie de même structure déjà existant.

1.2 Quels sont les problèmes qu'on doit résoudre pour obtenir ces résultats ?

1.2.1 Problème n°1 : créer un fichier à structure arborescente.

1.2.2 Problème n°2 : pouvoir continuer la création d'un fichier à structure arborescente.

1.2.3 Problème n°3 : pouvoir inclure dans le contenu d'un noeud d'un fichier à structure arborescente soit les informations qui constituent le contenu d'un noeud d'un autre fichier ayant une tell structure, soit des références à ces informations, références que le système pourra utiliser pour générer ces informations.

1.2.4 Problème n°4 : pouvoir désigner un noeud ou une famille de noeuds à l'aide d'un identifiant de noeud.

1.2.5 Problème n°5 : pouvoir créer un nouveau noeud comme étant le fils aîné, le fils cadet ou le frère cadet d'un noeud ou d'une famille de noeuds appartenant à des fichiers à structure arborescente déjà existants.

1.2.6 Problème n°6 : pouvoir retrouver tous les noeuds du sous-arbre du (des) noeud (s) spécifié (s) par un identifiant de noeud à partir d'un numéro de niveau a jusqu'à un numéro de niveau b.

1.2.7 Problème n°7 : supprimer un noeud ou une famille de noeuds ainsi que tous leurs descendants des fichier à structure arborescente auxquels ils appartiennent.

1.2.8 Problème n°8 : compléter les contenus d'un noeud ou d'une famille de noeuds par un texte donné.

1.2.9 Problème n°9 : pouvoir modifier les noms et (ou) les contenus d'un noeud ou d'une famille de noeuds.

2. Analyse fonctionnelle des différents problèmes et solutions adoptées.

2.1 Conventions du langage.

Pour décrire la syntaxe de nos fonctions de traitement, nous allons utiliser la forme traditionnelle de Backus-Naur et adopter les conventions suivantes :

⟨élément⟩ : élément du langage à définir par une règle

::= : indique que l'élément à définir à gauche est défini par l'expression de droite. Le tout étant une règle.

⟨élément 1⟩ } ⟨élément 2⟩|...|⟨élément n⟩ : l'un des éléments doit apparaître.

$\{\langle\text{élément}\rangle\}_0^1$: l'élément doit apparaître de 0 à 1 fois.

{⟨élément⟩} : l'élément doit apparaître un nombre quelconque de fois y compris zéro.

2.2 Problème n°1 : Créer un fichier à structure arborescente.

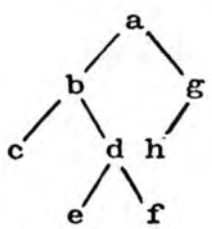
2.2.1 Solution a

Pour créer une structure d'arbre, on a besoin d'une fonction de création : CREATE.
Tous les noeuds étant parfaitement identifiés par le chemin qui les relie à la racine, une première solution sera de recopier dans la commande de création le nom des différents noeuds qui se trouvent sur le chemin qui relie le noeud à créer à la racine.
Notre fonction create pourrait avoir la syntaxe suivante

/ C {⟨nom de noeud⟩.} ⟨nom du nouveau noeud⟩

Exemple

Soit à créer la structure arborescente suivante :



On aura la séquence de commandes suivante :

/ C a

/ C a.b

/ C a.b.c

/ C a.b.d

/ C a.b.d.e

/ C a.b.d.f

/ C a.g

/ C a.g.h

L'ordre d'écriture de ces commandes peut être tout à fait quelconque puisque la syntaxe de la fonction CREATE nous rappelle à tout moment le nom du fichier (nom de la racine) dans lequel on désire créer un noeud ainsi que le chemin qui relie ce noeud au sommet de l'arbre.

On peut donc utiliser cette technique pour un parcours quelconque qui ne créerait pas un fils avant son frère aîné ni avant son père. Mais la lourdeur de l'écriture nous incite à choisir un ordre plus spécifique : les noeuds seront créés dans l'ordre où ils apparaissent dans le parcours dynastique.

2.2.2 Solution b

Lorsque le nouveau noeud qu'on désire créer est aussi le fils aîné du noeud créé précédemment, il devient parfaiteme inutile de récrire tout le chemin et la séquence de création de notre exemple peut être remplacée par la suivante :

/ C a

/ C b

/ C c

/ C a.b.d

/ C e

/ C a.b.d.f

/ C a.g

/ C h

Cette solution ne permet la simplification que d'un certain nombre de commandes CREATE puisqu'elle nous impose, lorsqu' on a terminé la création d'une feuille (c,e) ou d'un sous--arbre (f), de récrire tout le chemin qui relie la racine

au nouveau noeud qu'on désire créer.
Pour obtenir une écriture homogène : / C < nom de noeud >, on devrait pouvoir disposer d'une nouvelle commande qui signalerait qu'on vient de terminer la création d'une feuille ou d'un sous-arbre.

2.2.3 Solution c

Nous allons envisager une commande de terminaison END qui aura la syntaxe suivante : / E {< nom de noeud >}
A défaut du nom de noeud, cette fonction signale qu'on vient de terminer la création d'une feuille dont le nom est spécifié dans le create précédent et qu'on s'attend à créer un nouveau noeud comme frère cadet du noeud créé précédemment.

Si le nom de noeud est spécifié, cela signifie qu'on vient de terminer la création d' un sous-arbre dont la racine est ce nom de noeud et qu'on s'attend à créer un nouveau noeud comme frère cadet de ce dernier.

Si le nom de noeud mentionné dans le END est le nom du fichier (nom de la racine), cela signifie qu'on a terminé la création de la structure arborescente.

Pour créer la structure arborescente de l'exemple du point 2.2.1 la séquence de commandes sera :

```
/ C a
/ C b
/ C c
/ E
/ C d
/ C e
/ E
/ C f
/ E b
/ C g
/ C h
/ E a
```

Remarque

Au lieu de conclure notre séquence de commandes par / E a, on aurait pu écrire deux / E successifs et terminer notre séquence par

```
/ C h
/ E
/ E
```

2.3 Problème n°2 : pouvoir continuer la création d'un fichier à structure arborescente en ajoutant un sous-arbre au dernier noeud créé dans le parcours dynastique.

Nous avons vu au problème n°1 que pour créer un fichier à structure arborescente, il suffisait d'écrire une séquence de commandes du type Create et End ; la première commande devant nécessairement être un / C < nom du fichier >
Lorsqu'on désire reprendre un fichier pour continuer la création de noeuds en suivant le parcours dynastique, il est important de connaître le nom de ce fichier et de pouvoir retrouver le nom du dernier noeud créé.
Les informations fournies par la commande / C < nom de noeu étant insuffisantes, nous allons y ajouter le nom du fichier.
La syntaxe de notre commande Create aura l'allure suivante

$$/ \quad C \quad \left\{ \langle \text{nom de fichier} \rangle \right\}_{0}^{1} \quad \langle \text{nom de noeud} \rangle$$

La première commande de création de la nouvelle séquence devra nécessairement mentionner le nom du fichier utilisé.

2.4 Problème n°3 : pouvoir inclure dans le contenu d'un noeud d'un fichier à structure arborescente soit les informations qui constituent le contenu du noeud d'un autre fichier ayant une telle structure, soit des références à ces informations, références que le système pourra utiliser pour générer ces informations.

Pour résoudre ce problème, il suffit de se poser la question suivante :

Comment distinguer le cas où on désire que le contenu du nouveau noeud contienne des références à des informations pré-enregistrées de celui où on désire que ces références soit effectivement remplacées par les informations référencées ?

La solution c'est d'offrir à l'utilisateur la possibilité de la signaler lors de la commande de création à l'aide de l'élément <fonction> qui prendra soit la valeur < V > soit la valeur < R > ; < V > est synonyme de "prendre la valeur" et < R > est synonyme de "laisser tel quel".
Si le symbole < V > est présent, cela signifie que l'éditeur devra remplacer toutes les références à des informations pré-enregistrées qu'il rencontrera dans le contenu associé au nouveau noeud par le contenu des noeuds qui seront identifiés par ces références.
Si le symbole < R > est présent, l'éditeur devra considérer les références éventuelles qu'il rencontrera dans le contenu du nouveau noeud comme faisant partie de ce contenu et ne devra pas les traiter.

2.5 Syntaxe de la commande CREATE.

/ C ⟨fonction⟩, { (⟨nom de fichier⟩) } 1 0 ⟨nom de noeud⟩ { "⟨contenu⟩" }

⟨nom de fichier⟩ ::= ⟨nom de noeud⟩

⟨nom de noeud⟩ ::= ⟨lettre⟩ { ⟨lettre ou chiffre⟩ }

⟨lettre ou chiffre⟩ ::= ⟨lettre⟩ | ⟨chiffre⟩

⟨fonction⟩ ::= ⟨V⟩ | ⟨R⟩

⟨contenu⟩ ::= texte structuré en lignes

2.6 Syntaxe de la commande END

/ E { ⟨nom de noeud⟩ }

2.7 Problème n°4 : pouvoir désigner un noeud ou une famille de noeuds à l'aide d'un identifiant de noeud.

Lorsqu'on réalise une application informatique, on constate que si plusieurs routines sont fréquemment utilisées dans un grand nombre de programmes de l'application, il en est de même des enregistrements de plusieurs fichiers: fichier signalétique des clients, fichier des commandes du jour, fichier des produits, fichier des factures, etc.. Grâce à notre système, il devient donc parfaitement inutile de recopier ces enregistrements dans tous les programmes qui les utilisent : leur description détaillée peut être faite une seule fois et introduite dans tout programme à l'aide de notre éditeur. Dans l'optique de ce mémoire, ces différents fichiers seraient considérés comme des fichiers à structure arborescente dont les contenus des différents noeuds contiendraient par exemple :

- une description générale du contenu du fichier et son utilité
- une description générale de l'organisation du fichier (séquentielle, directe, indexée séquentielle..
- une description du format des enregistrements du fichier
- le fichier des données proprement dites
- etc...

Pour que notre éditeur puisse introduire dans un programme de l'application la description du format des enregistrements du fichier signalétique des clients par exemple, il faut qu'il dispose des renseignements nécessaires (identifiant de noeud) pour identifier le noeud en question.

2.7.1 Le problème qui se pose est donc le suivant :

Comment désigner un noeud à l'aide d'un identifiant de noeud ?

i Solution a

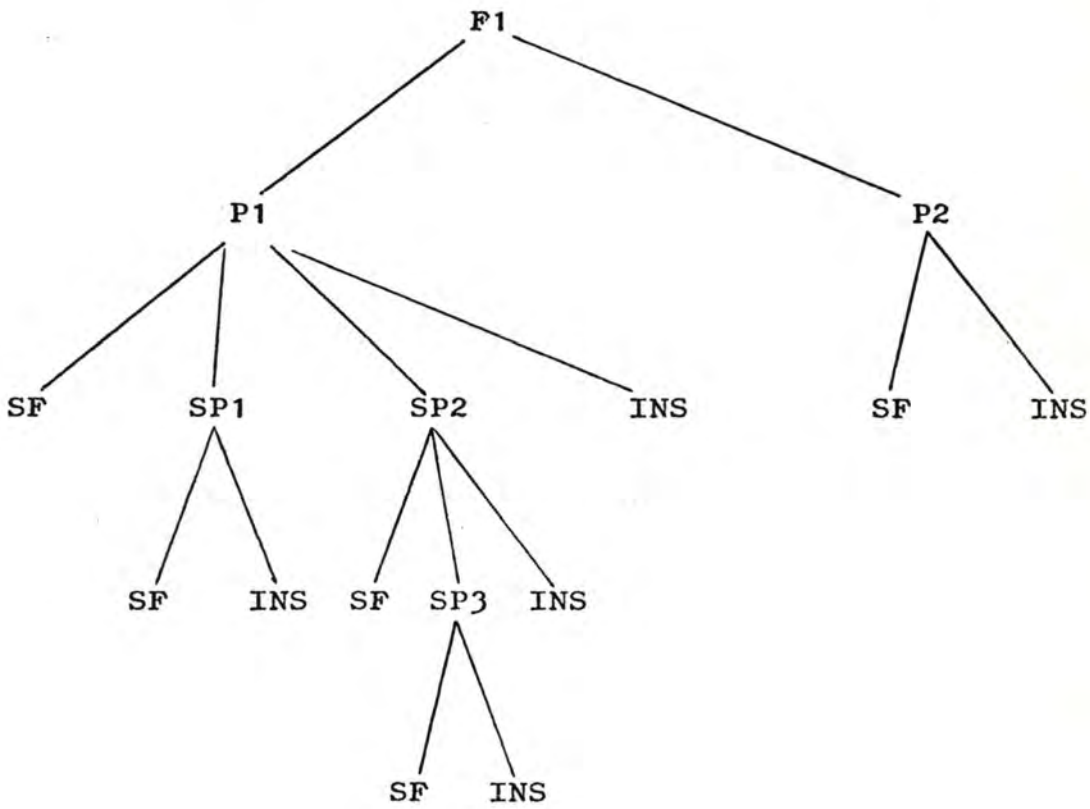
Puisque les noeuds sont caractérisés par le chemin qui les relie à la racine, une solution serait que ⟨identifiant de noeud⟩ reprenne en détail tout le chemin qui relie la racine au noeud dont on désire générer le contenu.

⟨identifiant de noeud⟩ aurait donc la syntaxe suivante:

⟨identifiant de noeud⟩::=⟨nom de noeud⟩{.⟨nom de noeud⟩}

ii Exemple

Considérons la structure arborescente suivante :



F1 est le nom du fichier sur lequel sont enregistrés les deux programmes principaux P1 et P2.
SP1; SP2 et SP3 désignent les sous-programmes du programme P1, SP3 désigne un sous-programme de SP2.
Le contenu d'un noeud de nom SF (spécifications fonctionnelles) contient les commentaires relatifs aux spécifications fonctionnelles d'un programme ou d'un sous-programme.

Le contenu d'un noeud de nom INS (instructions) contient

les instructions d'un programme ou d'un sous-programme.

Supposons que l'on désire générer les spécifications fonctionnelles du sous-programme SP3.
Notre solution a nous impose d'écrire comme identifiant du noeud désiré l'expression : F1.P1.SP2.SP3. SF

iii Solution b

Si nous reprenons notre structure arborescente, nous constatons qu'il n'y aurait aucune ambiguïté à utiliser comme identifiant du noeud désiré l'une des expressions suivantes :

```
F1.    .SP2.SP3.SF
F1.    .   .SP3.SF
F1.    .   .   .SF
  .SP1.SP2.SP3.SF
  .    .SP2.SP3.SF
  .    .   .SP3.SF
  .    .   .   .SF
```

Ce procédé nous permet de désigner les spécifications fonctionnelles du sous-programme SP3 d'une façon moins lourde :

F1....SF signifie qu'entre le noeud de nom F1 et le noeud de nom SF il y a 3 niveaux non spécifiés

....SF signifie qu'entre le noeud de nom SF et la racine il y a 3 niveaux non spécifiés.

Dans la pratique, il arrivera très souvent que l'utilisateur ne sache pas déterminer le niveau de la structure arborescente où se situe les spécifications fonctionnelles du sous-programme SP3.

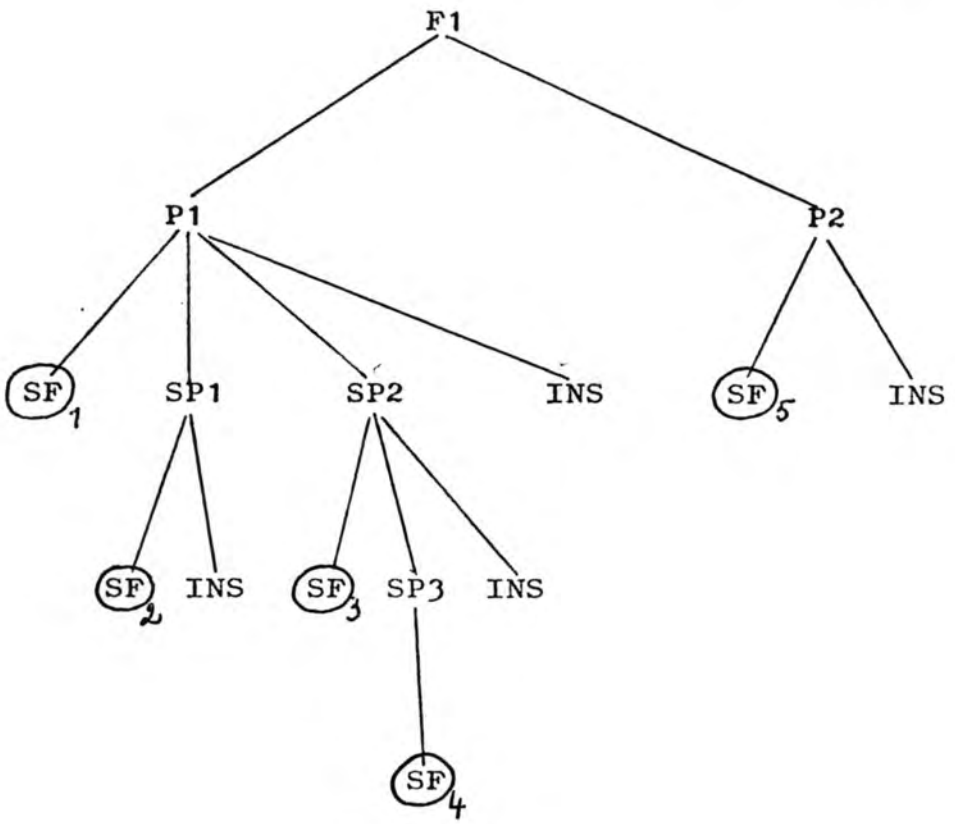
Si nous adoptons comme convention que le symbole * représente un nombre quelconque de niveaux non spécifiés(y compris 0); l'expression ...SP3.SF pourrait s'écrire*SP3.SF sans ambiguïté.

2.7.2 Pouvoir désigner une famille de noeuds à l'aide d'un identifiant de noeud:

En simplifiant la syntaxe de notre identifiant de noeud , nous avons offert à l'utilisateur la possibilité de désigner une famille de noeuds à l'aide d'un seul identifiant de noeud.
En effet, si l'on désire obtenir toutes les spécifications fonctionnelles se trouvant dans le fichier F1, il suffirait d'écrire : F1*SF.
De même les expressions F1.P1..SF ou F1...SF ou ...SF nous permettraient d'obtenir les spécifications fonctionnelles des sous-programmes SP1 et SP2; ce que l'on pourrait écrire mieux encore *P1..SF

2.7.3. Exemples

i. Considérons la structure arborescente suivante



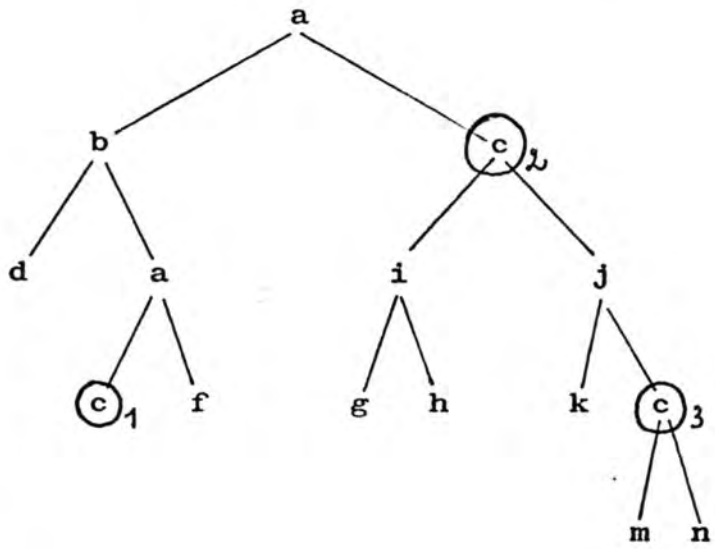
⟨identifiant de noeud⟩	⟨noeuds désignés⟩
F1.P1.SF	SF_1
F1. .SF	SF_1 et SF_5
F1.P1*SF	SF_1, SF_2, SF_3, SF_4
*P1..SF	SF_2, SF_3
F1.P1...SF	SF_4
F1..SP2.SF	SF_3
*SP1.SF	SF_2
F1.P1*..SF	SF_2, SF_3, SF_4
*SF	$SF_1, SF_2, SF_3, SF_4, SF_5$

ii. Explications

*P1..SF désigne la famille des noeuds de nom SF comme étant les petits-fils des noeuds de nom P1, ces noeuds de nom P1 pouvant se trouver n'importe où dans la structure d'arbre.

F1.P1*..SF désigne la famille des noeuds de nom SF comme étant les petits-fils ou les arrière-petits-fils,etc... des noeuds de nom P1 qui sont les fils de la racine de nom F1.

iii. Considérons la structure arborescente suivante



⟨identifiant de noeud⟩	⟨noeuds désignés⟩
a.c	c_2
*a.c	c_1 et c_2

2.7.4 Syntaxe d'un identifiant de noeud

⟨identifiaht de noeud⟩::=$\left\{\langle\text{nom de la racine}\rangle\right\}_0^1$⟨queue d'identifiant⟩

⟨queue d'identifiant⟩::=⟨séparateur⟩⟨nom de noeud⟩ | ⟨queue d'identifiant⟩⟨séparateur⟩⟨nom de noeud⟩

⟨séparateur⟩::= *{.} | .{.}

2.7.5 Remarque

L'intérêt de cette syntaxe est qu'elle donne à l'utilisateur le moyen de désigner une famille de noeuds à l'aide d'un seul identifiant de noeud. Il serait donc tout à fait possible que l'utilisateur s'attendant à obtenir le contenu d'un noeud unique en obtienne effectivement plusieurs. Pour éliminer ce danger, nous allons introduire un nouveau symbole : ! qui permettra à l'utilisateur de signaler dans les commandes qui utilisent un identifiant de noeud qu'il s'attend à désigner un et un seul noeud.
Si le symbole ! est présent cela signifie que l'utilisateur s'attend à ce que le système identifie un seul et unique noeud à l'aide de l'identifiant de noeud.
Si le système en trouve plusieurs, l'utilisateur désire recevoir un message d'erreur.
Si le symbole ! est absent, il n'y aura pas de message d'erreur puisque le système devra identifier un ou plusieurs noeuds à l'aide de l'identifiant de noeud.

2.8 Problème n°5 : Pouvoir créer un nouveau noeud comme étant le fils aîné, le fils cadet ou le frère cadet d'une famille de noeuds appartenant à des fichiers à structure arborescente, déjà existants.

Au stade actuel de l'analyse de ces problèmes, la seule fonction de création disponible est la fonction create qui nous permet de créer un fichier à structure arborescente en créant les différents noeuds dans l'ordre où ils apparaissent dans le parcours dynastique. Cette commande create ne nous permet pas de créer des noeuds individuellement ce qui restreint fortement les possibilités de notre éditeur : pour résoudre le problème n°6 il faudrait recréer tout le fichier.

Nous allons introduire une nouvelle fonction de création "la fonction INSERT " dont la syntaxe devra fournir au système le moyen de retrouver le noeud ou la famille de noeuds auxquels on désire ajouter un nouveau noeud et le lien de parenté qui doit les relier.

La syntaxe de cette nouvelle commande pourra donc s'écrire

$$/I\langle\text{fonction}\rangle\left(\left\{!\right\}_{0}^{1}\langle\text{identifiant de noeud}\rangle\right)\langle\text{lien de parenté}\rangle,$$

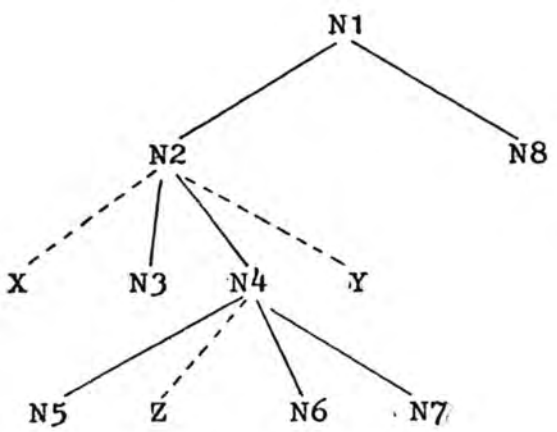
$$\langle\text{nom de noeud}\rangle\left\{"\langle\text{contenu}\rangle"\right\}_{0}^{1}$$

⟨lien de parenté⟩ devra spécifier s'il s'agit d'un fils aîné, fils cadet ou frère cadet.

Pour simplifier la syntaxe nous utiliserons les symboles suivants :

YO (younger) pour fils cadet
EL (elder) pour fils aîné
YB (young brother) pour frère cadet.

Exemple Considérons la structure arborescente suivante :



Opérations	Commandes
-ajouter un fils aîné de nom X au noeud de nom N2	/I⟨fonction⟩(! N2)EL,X "⟨contenu⟩"
-ajouter un fils cadet de nom Y au noeud de nom N2	/I⟨fonction⟩(! N2)YO,Y "⟨contenu⟩"
-ajouter un frère cadet de nom Z au noeud de nom N5	/I⟨fonction⟩(! N5)YB,Z "⟨contenu⟩"

Signalons que pour créer le noeud de nom Y on aurait pu utiliser la commande :

/I⟨fonction⟩(! N4)YB,Y "⟨contenu⟩"

2.9. Syntaxe de commande INSERT

/I⟨fonction⟩({ ! }$_0^1$ ⟨identifiant de noeud⟩) ⟨lien de parenté⟩,

⟨nom de noeud⟩ { "⟨contenu⟩" }$_0^1$

⟨lien de parenté⟩::=⟨fils aîné⟩ | ⟨fils cadet⟩ | ⟨frère cadet⟩

⟨fils aîné⟩::= EL

⟨fils cadet⟩::= YO

⟨frère cadet⟩::= YB

2.10 Problème n°6

Pouvoir retrouver tous les noeuds du sous-arbre du (des) noeuds (s) spécifiés par un identifiant de noeud à partir du numéro de niveau a jusqu'à un numéro de niveau b .

Lors du problème n°4 nous avons montré l'utilité de disposer d'un éditeur capable d'insérer un texte désigné par un nom dans un autre texte.

Pour réutiliser les informations appartenant aux contenus d'un noeud ou d'une famille de noeuds d'un ou plusieurs fichiers à structure arborescente, nous avons besoin d'une fonction de génération (la commande GENERATE) dont la syntaxe devra fournir au système le moyen d'identifier les différents noeuds :

/G { ! }$_0^1$ ⟨identifiant de noeud⟩

Le champ d'action de cette fonction est assez restreint puisqu'il permet uniquement de recopier les contenus propres des noeuds identifiés.

Dans cette optique, l'utilisateur qui désire générer les contenus de tous les noeuds faisant partie du sous-arbre spécifié par l'identifiant de noeud devra écrire autant de désignations qu'il y a de noeuds dans le sous-arbre en question. Il en est de même lorsque l'utilisateur désire générer les contenus de tous les noeuds d'un sous-arbre du niveau a au niveau b , les niveaux étant comptés relativement au (x) noeuds (s) désigné (s).

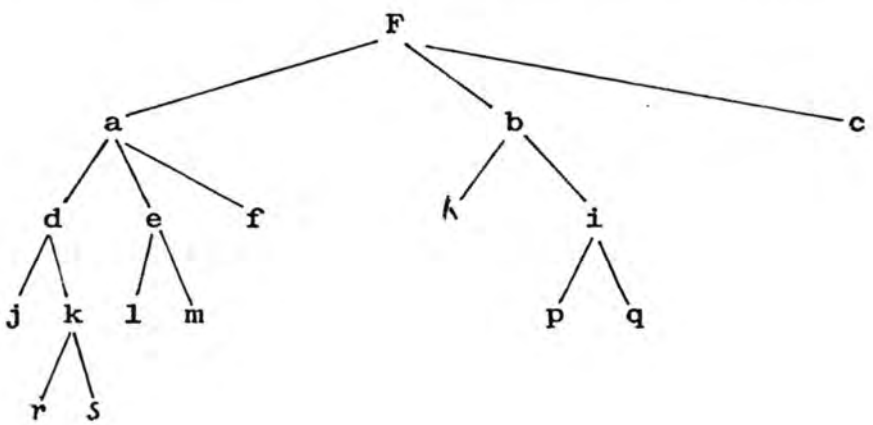
Pour supprimer cet inconvénient, nous allons compléter la syntaxe de la commande GENERATE :

$$/G \langle \text{argument} \rangle \left(\left\{ ! \right\}_{0}^{1} \langle \text{identifiant de noeud} \rangle \right)$$

où ⟨argument⟩ devra spécifier un intervalle de numéros de niveaux : entier 1,entier 2.

Ainsi /G 3,4 (! ⟨identifiant de noeud⟩) signifie qu'on désire générer le contenu de tous les noeuds du sous-arbre du noeud spécifié par l'identifiant de noeud à partir du niveau 3 et jusqu'au niveau 4 par rapport au noeud spécifié.

i. Exemple Considérons la structure arborescente suivante :



Commandes "generate"	Nom des noeuds dont le système génère le contenu.								
/ G 0,0 a	a								
/ G 0,∞ a	a	d	j	k	R	S	e	l	m
/ G 1,∞ a		d	j	k	R	S	e	l	m
/ G 0,1 a	a	d					e		
/ G 1,1 a		d					e		
/ G 3,3 a					R	S			
/ G 1,2 a		d	j	k			e	l	m

ii. Notation simplifiée.

Nous allons introduire un nouveau symbole ω et admettre la possibilité d'admettre entier 1 ou entier 2.
Si le symbole ω est absent, cela signifiera qu'on désire générer uniquement le contenu du nom de noeud spécifié par l'identifiant de noeud.
Si le symbole ω est présent, cela signifie qu'on désire générer le contenu de 1 ou plusieurs noeuds qui seront identifiés par l'identifiant de noeud et la valeur de l'argument.
Par défaut, entier 1 est égal à 0 et entier 2 est égal à ∞.
La séquence de commandes generate de l'exemple pourra se simplifier de la façon suivante :

/ G (a)

/ G ω (a)

/ G ω_1 (a)

/ G $\omega_{,1}$ (a)

/ G $\omega_{1,1}$ (a)

/ G $\omega_{3,3}$ (a)

/ G $\omega_{1,2}$ (a)

2.11. Syntaxe de la commande generate

/ G ⟨argument⟩ $\left(\left\{!\right\}_0^1 \langle \text{identifiant de noeud} \rangle\right)$

⟨argument⟩ ::= $\left\{\omega\left\{\left\{\text{entier}\right\}_0^1\left\{,\text{entier}\right\}_0^1\right\}_0^1\right\}_0^1$

⟨entier⟩ ::= ⟨chiffre⟩ | ⟨chiffre⟩ ⟨entier⟩

⟨identifiant de noeud⟩ ::= $\left\{\langle \text{nom de la racine} \rangle\right\}_0^1$ ⟨queue d'identifiant⟩

⟨queue d'identifiant⟩ ::= ⟨séparateur⟩ ⟨nom de noeud⟩ | ⟨queue d'identifiant⟩ ⟨séparateur⟩ ⟨nom de noeud⟩

⟨séparateur⟩ ::= * $\{.\}$ | . $\{.\}$

2.12 Problème n°7 Supprimer un noeud ou une famille de noeuds ainsi que tous leurs descendants des fichier à structure arborescente auxquels ils appartiennent.

Solution : Une fonction DELETE dont la syntaxe serait :

/ D $\left(\left\{!\right\}_{0}^{1}$ ⟨identifiant de noeud⟩$\right)$.

2.13 Problème n°8 Compléter les contenus d'un noeud ou d'une famille de noeuds par un texte donné.

Solution : Une fonction ADD dont la syntaxe serait :

/ A $\left(\left\{!\right\}_{0}^{1}$ ⟨identifiant de noeud⟩$\right)$ "⟨contenu⟩"

Le système concatenera ⟨contenu⟩ aux contenus existants des noeuds identifiés par ⟨identifiant de noeud⟩

2.14 Problème n°9 Pouvoir modifier les noms et/ou les contenus d'un noeud ou d'une famille de noeuds.

Solution : Une fonction MODIFY dont la syntaxe serait :

/ M $\left(\left\{!\right\}_{0}^{1}$ ⟨identifiant de noeud⟩$\right)$⟨modification⟩

⟨modification⟩ ::= ⟨nom de noeud⟩ | "⟨contenu⟩" | ⟨nom de noeud⟩ "⟨contenu⟩"

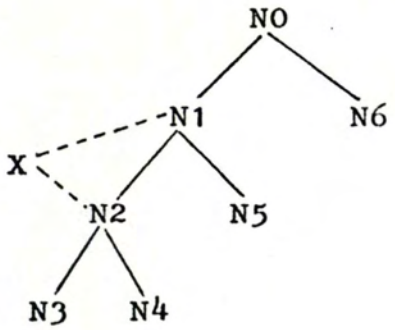
Si ⟨modification⟩::=⟨nom de noeud⟩ alors le système remplacera uniquement le nom du (des) noeud (s) référencé (s) par identifiant de noeud par la valeur de ⟨modification⟩

Si⟨modification⟩::= "⟨contenu⟩" seuls les contenus des noeuds références seront modifiés.

Dans le dernier cas, le système modifiera les noms et les contenus des noeuds référencés.

2.15 Problème non abordé Dans notre optique, notre système de gestion de fichiers à structure arborescente ne pourra pas résoudre le problème suivant :
Pouvoir créer un nouveau noeud comme étant à la fois le fils d'un noeud de nom N1 et le père d'un noeud de nom N2 ; le noeud de nom N1 étant le père du noeud de nom N2.

Exemple : Considérons la structure arborescente figurant à la page suivante :



Notre système ne nous permet pas de créer un noeud de nom X comme étant le fils du noeud de nom N1 et le père du noeud de nom N2.

---

3. Syntaxe des fonctions

Create

/ C ⟨destination⟩ $\left\{ \text{"⟨contenu⟩"} \right\}_0^1$

End

/ E $\left\{ \text{⟨nom de noeud⟩} \right\}_0^1$

Insert

/ I ⟨fonction⟩$\left(\left\{ ! \right\}_0^1 \text{⟨identifiant de noeud⟩} \right)$⟨lien de parenté⟩
⟨nom de noeud⟩$\left\{ \text{"⟨contenu⟩"} \right\}_0^1$

Generate

/ G ⟨argument⟩$\left(\left\{ ! \right\}_0^1 \text{⟨identifiant de noeud⟩} \right)$

Delete

/ D $\left(\left\{ ! \right\}_0^1 \text{⟨identifiant de noeud⟩} \right)$

Add

/ A $\left(\left\{ ! \right\}_0^1 \text{⟨identifiant de noeud⟩} \right)$ "⟨contenu⟩"

Modify

/ M $\left(\left\{ ! \right\}_0^1 \text{⟨identifiant de noeud⟩} \right)$⟨modification⟩

⟨destination⟩::=⟨fonction⟩,$\left\{ \left(\text{⟨nom de fichier⟩} \right) \right\}_0^1$⟨nom de noeud

⟨nom de fichier⟩::=⟨nom de noeud⟩

⟨fonction⟩::= ⟨V⟩|⟨R⟩

⟨nom de noeud⟩::=⟨lettre⟩$\left\{ \text{⟨lettre ou chiffre⟩} \right\}$

⟨lettre ou chiffre⟩::= ⟨lettre⟩|⟨chiffre⟩

⟨contenu⟩::= texte structuré en lignes à l'aide du caractère eoln qui indique la fin d'une ligne et du caractère etx qui indique la fin du texte. Le texte peut contenir des appels à l'éditeur au moyen de la fonction generate.

⟨identifiant de noeud⟩::=$\left\{ \text{⟨nom de la racine⟩} \right\}_0^1$⟨queue d'identifiant⟩

⟨queue d'identifiant⟩::= ⟨séparateur⟩⟨nom de noeud⟩|⟨queue

d'identifiant>⟨séparateur⟩⟨nom de noeud⟩

⟨séparateur⟩ ::= *{.}| .{.}

⟨lien de parenté⟩ ::= ⟨fils aîné⟩|⟨fils cadet⟩|⟨frère cadet⟩

⟨fils aîné⟩::= EL

⟨fils cadet⟩::= YO

⟨frère cadet⟩::= YB

⟨argument⟩::= $\left\{\omega\left\{\left\{\text{entier}\right\}_{0}^{1}\left\{,\ \text{entier}\right\}_{0}^{1}\right\}_{0}^{1}\right\}_{0}^{1}$

⟨entier⟩ ::= ⟨chiffre⟩|⟨chiffre⟩⟨entier⟩ .

⟨modification⟩::= ⟨nom de noeud⟩|"⟨contenu⟩"|⟨nom de noeud⟩ "⟨contenu⟩"

DEUXIEME PARTIE : ANALYSE ORGANIQUE

Deuxième partie : Analyse organique

Cette partie sera divisée en trois chapitres.

Dans le premier chapitre, nous aborderons tout d'abord le problème de la représentation en mémoire centrale d' une structure d'arbre et nous terminerons par une description du contenu de nos fichiers à structure arborescente.

Le deuxième chapitre sera consacré à la description des structures logiques des données. Chaque donnée sera désignée par un nom univoque qui sera utilisé par toute l'Analyse organique en guise d'identificateur. Ce nom conventionnel sera accompagné d'un libellé en clair mis entre parenthèses. Pour la description proprement dite de la structure nous utiliserons les conventions du langage de programmation Pascal.

Le troisième chapitre sera réservé à l'Analyse organique des traitements.

Chapitre I : Description du contenu des fichiers à structure arborescente.

1. Représentation d'une structure arborescente

Nous disposons d'une structure arborescente dont les différents noeuds possèdent un nom et un contenu. Si nous pouvons imposer au départ une taille maximum pour chaque nom de noeud, il n'en sera pas de même pour les contenus. D'autre part, comme ces contenus seront fort nombreux, il serait illogique de les garder en mémoire centrale. Une bonne solution pour représenter notre structure d'arbre en mémoire centrale, serait donc de constituer une table des noeuds avec pour les contenus un pointeur vers le premier enregistrement physique sur disque où commence le contenu d'un noeud.

2. Structure logique de la table des noeuds

2.1. Rappel des problèmes

2.1.1. Problème a

Pouvoir créer une structure arborescente en suivant le parcours dynastique.

2.1.2. Problème b

A partir d'un identifiant de noeud "$d_{0,1}\ n_1\ d_{1,2}\ n_2 \ldots n_{k-1}\ d_{k-1,k}\ n_k$" ($d_{ij}$ représente le nombre de niveaux séparant le noeud de nom n_i du noeud de nom n_j), retrouver le noeud ou la famille de noeuds désignés par cet identifiant.

2.1.3. Problème c

Parmi tous les noeuds du sous-arbre du (des) noeud (s) désignés par un identifiant de noeud, retrouver ceux dont le numéro de niveau relatif est compris entre 2 nombres a et b.

2.1.4. Problème d

Pouvoir ajouter un fils aîné à tous les noeuds d'une structure arborescente désignés par un identifiant de noeud.

2.1.5. Problème e

Pouvoir ajouter un fils cadet (ou un frère cadet) à tous les noeuds d'une structure arborescente désignés par un identifiant de noeud.

2.1.6. Problème f

Pouvoir supprimer le sous-arbre de tous les noeuds d'une structure arborescente désignés par un identifiant de noeud.

2.2 Recherche de la structure logique de la table des noeud

2.2.1. Solution a

Si nous analysons les énoncés des problèmes a,b,c,d, e,f, nous pouvons en conclure qu'il y a deux problèmes qui se rencontrent souvent :

i. Problème n°1

A partir d'un identifiant de noeud " $d_{0,1}\ n_1\ d_{1,2}\ n_2 \ldots n_{K-1}\ d_{K-1,K}\ n_K$ " retrouver le noeud ou la famille de noeuds désignés par cet identifiant.
L'algorithme de recherche sera très simple si on dispose pour chaque noeud d'un pointeur vers le père de ce noeud.

Pour résoudre ce problème, il suffit d'effectuer les opérations suivantes :

- parcourir séquentiellement la table des noeuds pour trouver un noeud de nom n_K
- utiliser les pointeurs pères pour remonter de $d_{K-1,K}$ niveaux
- vérifier si le nom du noeud trouvé dans la table est identique à n_{K-1}
- si les noms sont identiques, recommencer avec $d_{K-2,K-1}$, n_{K-2} et ainsi de suite jusqu'au moment où on a épuisé complètement la valeur de l'identifiant de noeud; dans le cas contraire, stopper la procédure de recherche pour ce noeud de nom n_K
- continuer le parcours séquentiel de la table des noeuds pour retrouver éventuellement un autre noeud de nom n_K et recommencer les opérations.

ii. Problème n°2

Pouvoir effectuer le parcours dynastique d'une structure d'arbre.

L'algorithme de parcours est immédiat si on dispose pour chaque nom de noeud d'un pointeur vers le noeud suivant dans le parcours dynastique.

iii. Solution

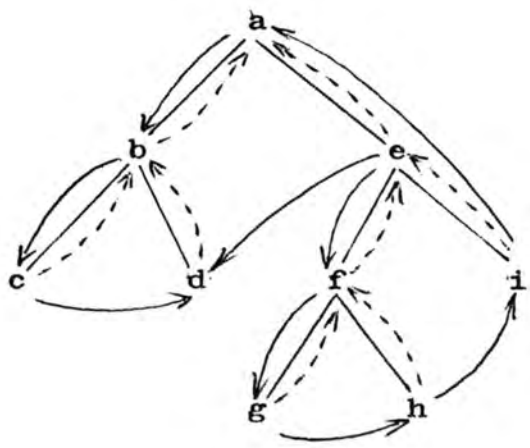
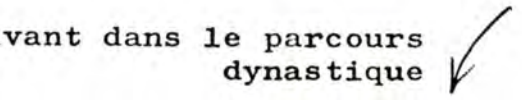


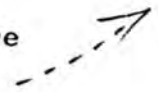
Table des noeuds

-nom du noeud

-pointeur vers le suivant dans le parcours dynastique



-pointeur vers le père



2.2.2. Solution b

Nous allons analyser la solution a en fonction des problèmes a,b,c,d,e,f.

i. Problème a

La résolution de ce problème n'offre aucune difficulté pour gérer nos 2 pointeurs, puisque nous disposons de la fonction create qui permet de créer un fichier à structure arborescente en suivant le parcours dynastique et la fonction end qui signale la fin de création d'une feuille ou d'un sous-arbre.

ii. Problème c et f

Pour résoudre ces 2 problèmes, le parcours dynastique devra reconstuire le numéro de niveau de chaque noeud :

- dans le cas du problème c pour retrouver tous les noeuds dont le numéro de niveau est compris entre 2 nombres a et b.
- dans le cas du problème f pour retrouver le suivant dans le parcours dynastique du dernier noeud du sous-arbre qu'on désire supprimer.
  Pour reconstruire le numéro de niveau, les algorithmes devront gérer une pile et effectuer des tests sur les pointeurs père. En plus de cette complexité le temps d'exécution de ces algorithmes risque d'être fort long : dans le cas du problème c par exemple, il faut effectuer le parcours dynastique complet du sous-arbre de chaque noeud désigné par l'identifiant de noeud.
  Nous pourrions diminuer considérablement soit la complexité, soit le temps d'exécution de ces algorithmes au prix d'une information supplémentaire. Si on connaissait le numéro de niveau de chaque noeud, la gestion d'une pile ne s'imposerait plus et les algorithmes seraient plus simples.
  Quant au temps d'exécution, il pourrait être raccourci si on disposait pour chaque noeud d'un pointeur vers son frère cadet : pour le problème c , par exemple, il suffirait, lorsqu'on est arrivé à un noeud de niveau b , de court-circuiter le parcours dynastique de ses descendants en utilisant le pointeur vers le frère cadet. Les algorithmes seront plus rapides mais aussi plus complexes car il faudra tester pour chaque noeud l'existence d'un frère cadet.

Comme ces 2 solutions sont aussi coûteuses l'une que l'autre en placemémoire, le choix de l'une d' entre elles ou(des deux) n'est qu'une question de préférence.
Nous avons opté pour le numéro de niveau à cause de la simplicité des algorithmes de résolution des problèmes.

iii. Problèmes d et e

Le problème d n'offre guère de difficultés puisque le fils aîné d'un noeud, quand il existe, est toujours son suivant dans le parcours dynastique.
L'algorithme de résolution du problème e sera aussi simple que le précédent ; il risque d'être parfois beaucoup plus long car il faudra toujours effectuer le parcours dynastique complet du sous-arbre du noeud auquel on désire ajouter un fils cadet.

iv. Solution

C'est la solution a avec le numéro de niveau comme information supplémentaire.
La table des noeuds contiendra donc :

- nom du noeud
- numéro de niveau
- pointeur vers le suivant dans le parcours dynastique
- pointeur vers le père
- pointeur vers le premier enregistrement physique sur disque où commence le contenu du noeud.

3. Description du contenu des fichiers à structure arborescente.

Convention : Les enregistrements physiques de ces différents fichiers ont une longueur fixe de 80 caractères.

3.1. Gestion de l'ensemble des enregistrements physiques disponibles d'un fichier.

3.1.1. 1ère méthode : Chaîner les enregistrements physiques

Chaque enregistrement physique contient une zone réservée à un pointeur. Avant la première utilisation du fichier, ou chaîne chaque enregistrement physique au suivant. Le dernier enregistrement contient dans la zone réservée au pointeur une indication spéciale de fin de chaîne. L'adresse du 1er enregistrement physique est placée dans une zone en mémoire centrale.

Avantages :
- cette méthode n'est pas coûteuse en place de mémoire centrale (une zone pour contenir l'adresse du premier enregistrement physique disponible).
- la programmation est relativement simple.

Inconvénients :
- on doit formatter l'ensemble des enregistrements physiques d'un fichier avant

sa première utilisation
- on doit réserver dans chaque enregistrement physique une zone pour contenir un pointeur

3.1.2. 2me méthode : dresser une image du support.

On réserve un nombre n d'enregistrements physiques au début de chaque fichier pour constituer un mapping des enregistrements physiques disponibles sur le support. Ce mapping est constitué par une table de bits à raison de 1 bit par enregistrement physique. Les bits correspondant aux zones occupées seront mis à 0 (false), ceux correspondant aux zones libres à 1 (true).

Avantages : Cette solution ne présente aucun des inconvénients cités plus haut.

Inconvénients :
- difficulté de programmation plus grande
- la place en mémoire centrale est légèrement plus coûteuse puisqu'une table de 150 bytes suffirait à gérer un fichier de 1200 enregistrements physiques.

3.1.3. Conclusion : Notre système de gestion de fichiers étant destiné à des micro-ordinateurs, nous utiliserons la deuxième méthode pour éviter une initialisation trop longue.

3.2. Description du contenu des enregistrements physiques.

- Les n premiers enregistrements physiques contiendront la table des bits constituant l'image du support ; n étant une constante du système.
- Le (n+1)ième enregistrement physique contiendra le début de la table des noeuds.
- Les enregistrements physiques restant seront alloués dynamiquement en fonction des places libres et contiendront les contenus des différents noeuds et la suite de la table des noeuds. Les enregistrements physiques contenant la table des noeuds devront être chaînés entre eux à l'aide d'un pointeur et le dernier de ces enregistrements contiendra dans la zone réservée au pointeur une indication de fin de table. Il en sera de même pour les enregistrements physiques du contenu de chaque noeud.

3.3. Notation

i. A chaque fichier est donc associé une table des noeuds de ce fichier, table que nous appellerons "table partielle des noeuds". En mémoire centrale se trouvera une table de tous les noeuds de tous les fichiers traités dans l'application, table que nous appellerons "table des noeuds"

ii. De la même façon, la table des places libres de chaque fichier sera appelée "table partielle des places libres", et la table en mémoire centrale qui contiendra l'ensemble de ces tables sera appelée "table des places libres."

iii. Le fichier d'entrée qui contiendra l'ensemble des commandes pour l'éditeur s'appellera F0.
Les fichiers à structure arborescente, traités dans l'application, seront désignés par F1,F2,F3 et F4.

CHAPITRE II : DESCRIPTION DES STRUCTURES LOGIQUES DES DONNEES.

La description d'une structure donnée comporte l'attribution d'un identificateur grâce auquel la structure pourra être référencée et la description proprement dite de la structure. Dans le dictionnaire des données, l'identificateur sera accompagné d'un libellé en clair mis entre parenthèses et d'une note explicative concernant la donnée proprement dite. Chaque identificateur sera composé de la concaténation d'un ou plusieurs codes mnémoniques de trois lettres; chacun de ces codes ayant une signification propre. Dans la partie réservée aux annexes, nous établirons un dictionnaire des codes mnémoniques. Le dictionnaire des données sera présenté de deux manières différentes :
- une présentation alphabétique (voir annexe B)
- une présentation logique.

1. Présentation logique du dictionnaire des données

1.1 Tableaux

1.1.1. Pile des adresses : ASK

i. ASK : array [1..ASKMXL] of integer

constante : ASKMXL
indice : ASKCRX
donnée de manoeuvre : ASKKPX

ii. ASK (stack of address) : array [1..ASKMXL] of integer
Cette pile contiendra les adresse logiques dans la table des noeuds de tous les noeuds de la famille de noeuds désignés par un identifiant de noeud.

ASKCRX (stack of address current index) : 0..ASKMXL
indice de la pile ASK

ASKKPX (stack of address keep index) : 0..ASKMXL
donnée de manoeuvre qui sert à mémoriser la hauteur dans la pile ASK du premier noeud de la famille de noeuds désignés par un identifiant.

ASKMXL (maxlength of ASK) : constante
taille maximum de la pile ASK

1.1.2. Tableau de bits : BIT

i. BIT : packed array [1..BITMXL] of boolean

constante : BITMXL

indice : BITCRX

ii. BIT (bit) : packed array [1..BITMXL] of boolean
Buffer de bits utilisé pour la lecture ou l'écriture des enregistrements physiques réservés aux tables partielles des places libres

BITCRX (bit current index) : 0..BITMXL
indice du tableau BIT

BITMXL (maxlength of BIT) : constante
taille maximum du tableau BIT

1.1.3. Nom du noeud courant : CRN

i. CRN : packed array [1..NNMMXL] of char

constante : NNMMXL

indice : CRNCRX

ii. CRN (current name) : packed array [1..NNMMXL] of char
donnée de manoeuvre contenant le nom du noeud courant dans la table des noeuds.

CRNCRX (current name current index) : 0..NNMMXL
indice du tableau CRN

NNMMXL (maxlength of node name) : constante
taille maximum d'un nom de noeud

1.1.4. Pile des noms de fichiers : FSK

i. FSK : array [1..FSKMXL] of FSKENT

FSKENT : packed array [1..NNMMXL] of char

constante : FSKMXL
NNMMXL

indice : FSKCRX

ii. FSK (stack of file name) : array [1..FSKMXL] of FSKENT
pile des noms de fichiers utilisés dans notre application. Ces noms sont considérés comme des châines de caractères; le langage PASCAL n'offrant pas la possibilité de manipuler des variables "nom de fichier".

FSKENT (entry of FSK) : packed array [1..NNMMXL] of char
élément de la pile FSK

FSKCRX (FSK current index) : 0..FSKMXL
indice de la pile FSK

FSKMXL (maxlength of FSK) : constante
taille maximum de la pile FSK
Elle est égale au nombre de fichiers utilisés dans notre application.

NNMMXL (maxlength of node name) : taille maximum d'un nom de noeud

1.1.5. Table des places libres : FTB

i. FTB : array [1..FTBMXL] of FPL

FPL : packed array [1..FPLMXL] of boolean

constantes : FTBMXL
FPLMXL

indices : FTBCRX : 0..FTBMXL
FPLCRX : 0..FPLMXL

données de manoeuvre : FTBKPX : 0..FTBMXL
FPLKPX : 0..FPLMXL

ii. FTB (freetable) : array [1..FTBMXL] of FPL
table des places libres constituée séquentiellement en mémoire centrale par l'ensemble des tables partielles des places libres de tous les fichiers traités dans notre application.

FTBCRX (freetable current index) : 0..FTBMXL
indice du tableau FTB

FTBKPX (freetable keep index) : 0..FTBMXL
donnée de manoeuvre qui mémorise la valeur de l'indice FTBCRX

FTBMXL (maxlength of FTB) : constante
taille maximum de la table des places libres.
Elle est égale au nombre de fichiers utilisés dans notre application.

FPL (free place) : packed array [1..FPLMXL] of boolean
composant de la donnée composée FTB.
tableau de bits contenant l'ensemble des BNB premiers enregistrements physiques d'un fichier à structure arborescente (mapping des places libres dans le fichier).

FPLCRX (free place current index) : 0..FPLMXL
indice du tableau FPL

FPLKPX (free place keep index) : 0..FPLMXL
donnée de manoeuvre qui mémorise la valeur de l'indice FPLCRX

FPLMXL (maxlength of FPL) : constante
taille maximum du tableau FPL

1.1.6. Table des noeuds : NTB

i. NTB : array [1..NTBMXL] of NTBENT

```
NTBENT : record NNM : packed array [1..NNMMXL] of char
                LNB : integer
                RTP : 0..NTBMXL
                NXP : 0..NTBMXL
                FAP : 0..NTBMXL
                RCP : integer
          end
```

constantes : NTBMXL
NNMMXL

indices : NTBCRX : 0..NTBMXL
NNMCRX : 0..NNMMXL

donnée de manoeuvre : NTBKPX : 0..NTBMXL

ii. FAP (father pointer) : 0..NTBMXL
composant de la donnée composée NTBENT
pointeur contenant l'adresse logiqu dans la table des noeuds du père d'un noeud

LNB (level number) : integrer
composant de la donnée composée NTBENT
Elle indique le numéro de niveau du noeud courant.

NNM (node name) : packed array [1..NNMMXL] of char
composant de la donnée composée NTBENT
Elle indique le nom du noeud courant

NNMCRX (node name current index) : 0..NNMMXL
indice du tableau NNM

NNMMXL (maxlength of NNM) : constante
taille maximum d'un nom de noeud

NTB (node table) : array [1..NTBMXL] of NTBENT
table des noeuds constituée séquentiellement en mémoire centrale par l'ensemble des tables partielles des noeuds de tous les fichiers de notre application.

NTBCRX (note table current index) : 0..NTBMXL
indice de la table des noeuds NTB

NTBENT (entry of node table) : record (NNM, LNB, RTP, NXP, FAP, RCP)
élément de la table des noeuds NTB

NTBKPX (node table keep index) : 0..NTBMXL
donnée de manoeuvre qui mémorise la valeur de l'indice NTBCRX

NTBMXL (maxlength of NTB) : constante
taille maximum de la table des noeuds NTB

NXP (pointer to next) : 0..NTBMXL
composant de NTBENT
pointeur vers le noeud suivant dans le parcours dynastique

RCP (record pointer) : integer
composant de la donnée composée NTBENT
pointeur vers le premier enregistrement physique du contenu d'un noeud

RTP (root pointer) : 0..NTBMXL
composant de la donnée composée NTBEN
pointeur contenant l'adresse logique dans la table des noeuds de la racine d'un noeud.

1.1.7. Record de sortie : REC

i. REC : record ODT : packed array [1..ODTMXL] of char
NXR : integer
end

constantes : RECMXL
ODTMXL

indice : ODTCRX

ii. NXR (next record) : integer
composant des données composées BUF et REC
pointeur vers l'enregistrement suivant du contenu d'un noeud

ODT (output data) : packed array [1..ODTMXL]of char
composant de la donnée composée REC

ODTCRX (output data current index) : 0..ODTMXL
indice du tableau ODT

ODTMXL (maxlength of ODT) : constante
taille maximum du tableau ODT

REC (record) : record (ODT, NXR)
buffer d'écriture

RECMXL (maxlength of REC) : constante
taille maximum de REC

1.1.8. Table des racines : RTB

i. RTB : array [1..RTBMXL] of RTBENT

RTBENT : record RNM : packed array [1..NNMMXL] of char
NTBPTR : 0..NTBMXL
FTBPTR : 0..FTBMXL
end

constantes : RTBMXL
NNMMXL

indices : RTBCRX : 0..RTBMXL
RNMCRX : 0..NNMMXL

donnée de manoeuvre : RTBKPX : 0..RTBMXL

ii. FTBPTR (pointer to FTB) : 0..FTBMXL
composant de la donnée composée RTBENT
pointeur indiquant l'adresse logique dans la table des places libres (FTB) de la table partiel le des places libres d'un fichie

NNMMXL (maxlength of NNM) : constante
taille maximum d'un nom de noeud.

NTBPTR (pointer to node table) : 0..NTBMXL
composant de la donnée composée RTBENT
pointeur qui indique l'adresse logique dans la table des noeuds du début de la table partielle des noeuds d'un fichier.

RNM (root name) : packed array |1..NNMMXL| of char
composant de la donnée composée RTBENT
donnée de manoeuvre contenant le nom de la racine d'une structure arborescent

RNMCRX (root name current index) : 0..RNMMXL
indice du tableau RNM

RTB (root table) : array [1..RTBMXL] of RTBENT
table des racines

RTBCRX (root table current index) : 0..RTBMXL
indice du tableau RTB

RTBENT (root table entry) : record (RNM, NTBPTR, FTBPTR)
élément du tableau RTB

RTBKPX (root table keep index) : 0..RTBMXL
donnée de manoeuvre qui mémorise la valeur de l'indice RTBCRX

RTBMXL (maxlength of root table) : constante
taille maximum de la table des racines (RTB)

1.1.9. Stack : STK

i. STK : array [1..STKMXL] of STKENT

```
STKENT : record  FNM : packed array [1..NNMMXL] of char
                 CRP : 0..IDTMXL
                 BUF : record IDT : packed array [1..IDTMXL
                                   of char
                              NXR : integer
                       end
          end
```

constantes : STKMXL
NNMMXL
IDTMXL

indices : STKCRX
FNMCRX
IDTCRX

ii. BUF (buffer) : record (IDT, NXR)
composant de la donnée composée STKENT
buffer de lecture et d'écriture des enregistrements physiques réservés aux tables partielles des places libres des fichiers

CRP (current position) : 0..IDTMXL
composant de la donnée composée STKENT
donnée de manoeuvre qui mémorise la valeur de l'indice IDTCRX du tableau IDT

FNM (file name) : packed array [1..NNMMXL]of char
composant de la donnée composée STKENT
donnée de manoeuvre qui permettra au système de déterminer le fichier dans lequel on désire lire

FNMCRX (file name current index) : 0..NNMMXL
indice du tableau FNM

IDT (input data) : packed array [1..IDTMXL] of char
composant de la donnée BUF

IDTCRX (input data current index) : 0..IDTMXL
indice du tableau IDT

IDTMXL (maxlength of IDT) : constante
taille maximum du tableau IDT

NNMMXL : (maxlength of NNM) : constante
taille maximum d'un nom de noeud

NXR (next record) : integer
composant des données composées BUF et REC
pointeur vers l'enregistrement suivant du contenu d'un noeud

STK (stack) : array [1..STKMXL] of STKENT
stack utilisé pour la récursivité des commandes générale

STKCRX (stack current index) : 0..STKMXL
indice de STK

STKENT (stack entry) : record (FNM, CRP, BUF)
élément de STK

STKMXL (maxlength of STK) : constante
taille maximum du stack

1.1.10. Tableaux de transition : TRSARG et TRSNID
et tableaux d'actions associés : ACTARG et ACTNID

i. Tableau de transition d'un argument
TRSARG : array [0..4, 1..5] of integer

Tableau de transition d'un identifiant de noeud
TRSNID : array [0..4, 1..6] of integer

Tableau d'actions d'un argument
ACTARG : array [0..4, 1..5] of procedure

Tableau d'actions d'un identifiant de noeud
ACTNID : array [0..4, 1..6] of procedure

ii. ACTARG (argument action table) : array [0..4,1..5] of procedure
tableau de procédures associé au tableau TRSARG

ACTNID (node identifier action table) : array [0..4, 1..6] of procedure
tableau de procédures associé au tableau TRSNID

TRSARG (argument transition table) : array [0..4, 1..5] of integer
tableau de transition correspondant à la partie < argument d'une commande générale

TRSNID (node identifier transition table) : array [0..4, 1..6] of integer
tableau de transition correspondant à un identifiant de noeud

1.2. Autres données simples

BNB (bit number) : constante
nombre maximum d'enregistrements physiques réservés au début de chaque fichier pour contenir la table partielle des places libres de ce fichier

CRTLEV (current level) : integer
donnée de manoeuvre dont le contenu est égal au numéro de niveau plus 1 du dernier noeud créé dans une structure arborescente en suivant le parcours dynastique

CRTCAR (current character) : char
donnée de manoeuvre contenant le caractère courant

CRTNXR (current next record) : integer
donnée de manoeuvre qui mémorise la valeur du pointeur NXR de REC

LND (last node) : 0..NTBMXL
donnée de manoeuvre contenant l'adresse logique dans la table des noeuds du dernier noeud créé dans un fichier à structure arborescente en suivant le parcours dynastique

MNY (many) : boolean
variable booléenne qui prendra la valeur true lorsqu'un identifiant de noeud désignera une famille de noeuds

NNB (number of nodes) : integer
donnée de manoeuvre qui indique le nombre de noeuds identifiés par un identifiant de noeud

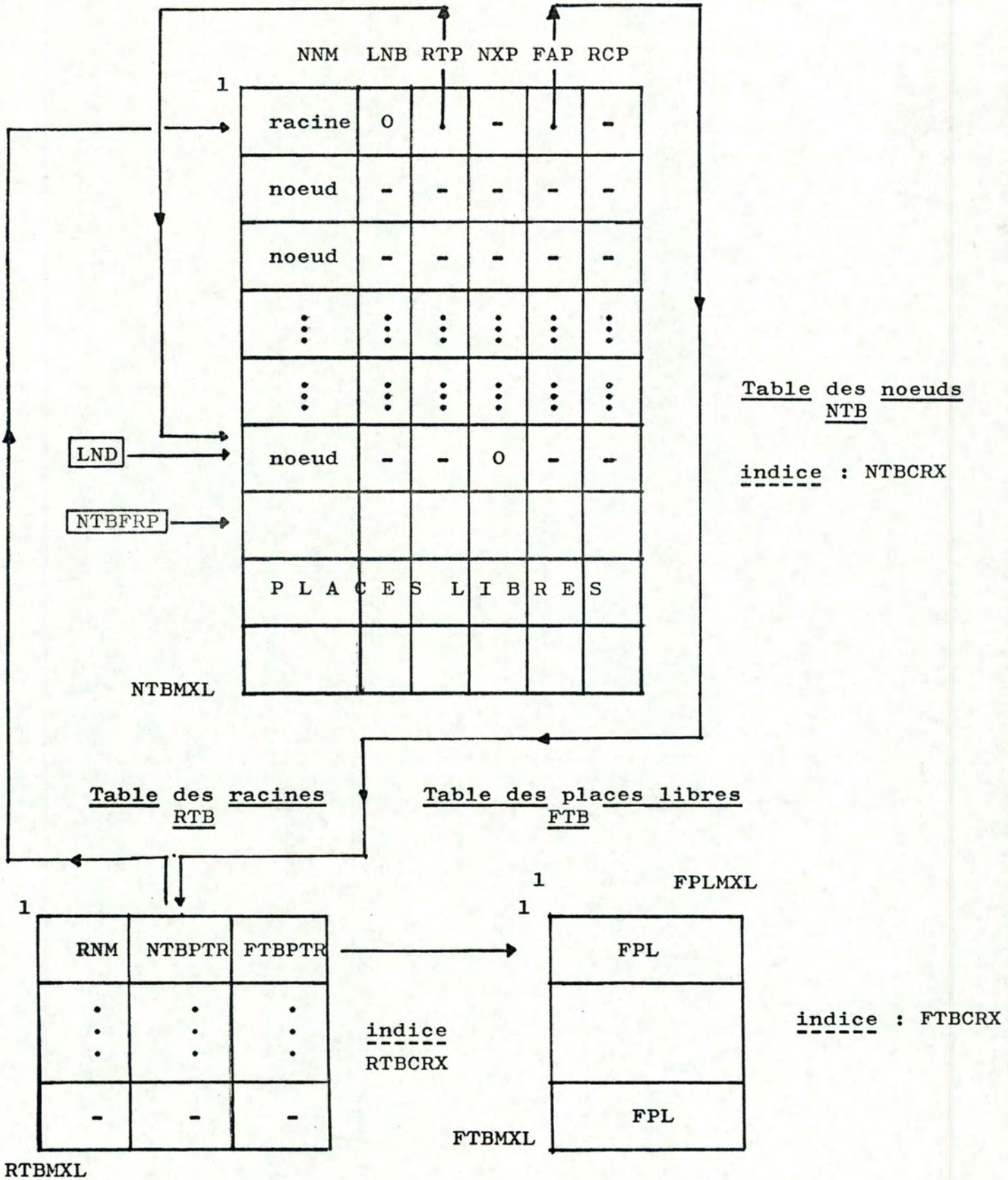
NSE (number of stack entry) : constante
nombre maximum d'éléments STKENT que peut contenir un enregistrement physiqu

NTBFRP (node table free pointer) : 0..NTBMXL
pointeur qui indique la première place libre disponible dans la table des noeuds NTB

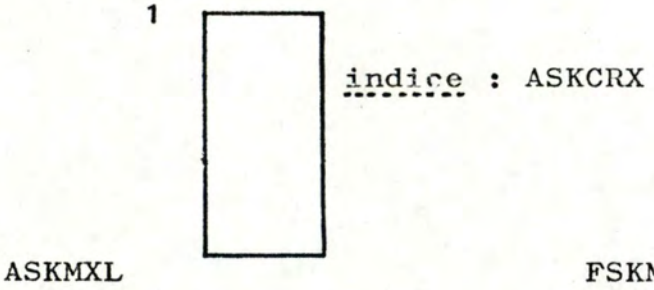
RNB (record number) : integer
donnée de manoeuvre qui sert à compter le nombre d'enregistrements physiques du contenu d'un noeud

VRF-SW (value or reference switch) : boolean
variable booléenne qui prendra la valeur true lorsque l'élément ‹ fonction › de la syntaxe des commandes Create et Insert aura la valeur 'V'.

2. Schéma des structures logiques des données



Pile des adresses ASK



Pile des noms de fichiers FSK

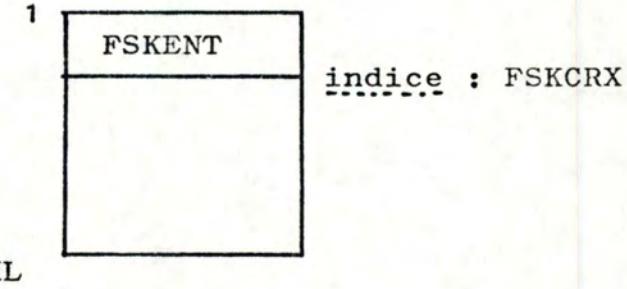
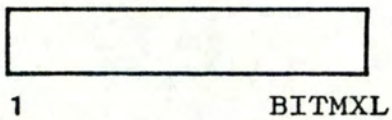
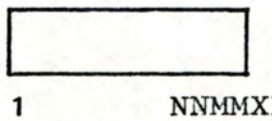


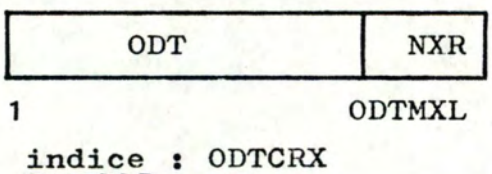
Tableau de bits BIT



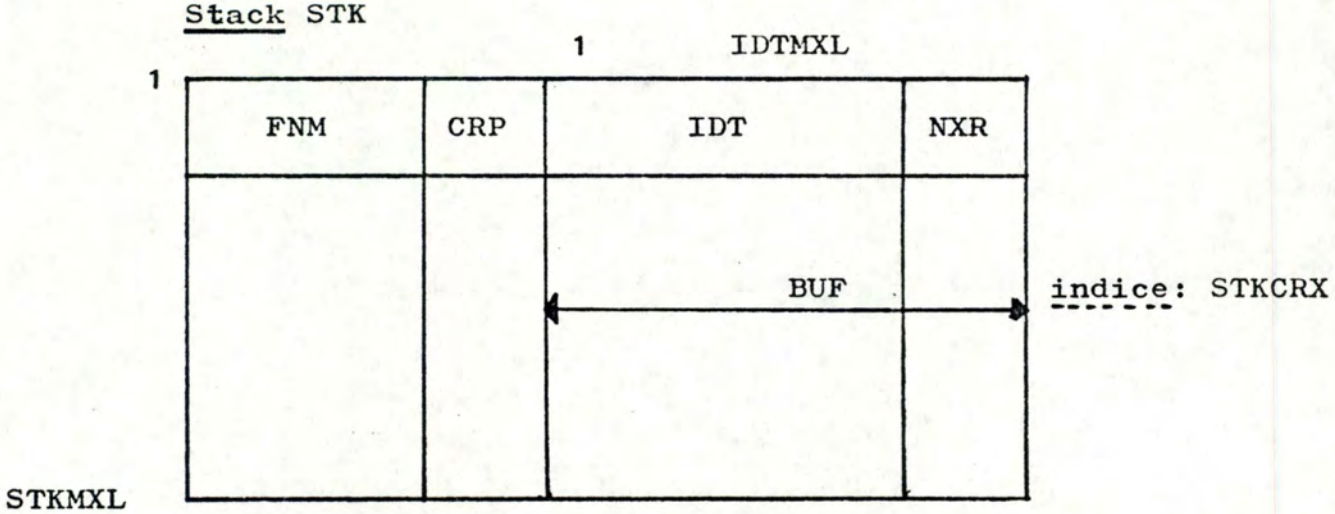
Nom du noeud courant CRN



Record de sortie : REC



Stack STK



CHAPITRE III : ANALYSE ORGANIQUE DES TRAITEMENTS

Ce chapître sera décomposé en cinq paragraphes :

- Conventions
- Déclaration
- Initialisation
- Iteration
- Terminaison

Dans le premier paragraphe, nous énoncerons les conventions de notre analyse organique.
Le deuxième paragraphe sera consacré à la déclaration de certaines procédures globales du programme; les structures de données,décrites dans le dictionnaire de données, ne seront plus déclarées.
Le troisième paragraphe sera consacré au chargement des tables partielles des places libres et des tables partielles de noeuds de tous les fichiers à structure arborescente traités dans notre application.
Le dernier paragraphe sera consacré au chargement de toutes les tables en mémoire secondaire.

PARAGRAPHE A : CONVENTIONS

Pour toute l'analyse organique des traitements, nous adopterons les conventions suivantes :

1. La désignation d'une procédure comportera l'attribution d'un identificateur (concaténation d'un ou plusieurs codes mnémoniques de trois lettres). Dans le texte de la procédure, nous décrirons dans une section 'Comment' les spécifications fonctionnelles de la procédure et éventuellement la signification de l'identificateur.

2. Chaque appel de procédure sera écrit en majuscules.

3. Nous disposons des procédures suivantes :

   - procedure READ (FO,enr)

     Enr est une zone en mémoire centrale de 80 caractères. Cette procédure lit un enregistrement physique du fichier séquentiel FO et le place en mémoire centrale dans la zone 'enr'.

   - procedure READ (F_i,x,enr) i : 1..4

     X est l'adresse sur disque d'un enregistrement physique du fichier F_i. Cette procédure lit dans le fichier F_i l'enregistrement physique qui se trouve à l'adresse 'x' et le place en mémoire centrale dans la zone 'enr'.

   - procedure WRITE (F_i,x,enr) i : 1..4

     Cette procédure écrit le contenu de la zone mémoire 'enr' dans le fichier F_i à l'adresse 'x'.

4. Toutes les commandes du fichier FO doivent nécessairement commencer en début de ligne à l'exeption des commandes GENERATE incorporées dans le texte du contenu du noeud.

5. Lors du chargement en mémoire centrale des tables des places libres et des tables partielles des noeuds des différents fichiers, nous construirons une table qui contiendra, pour chaque fichier à structure arborescente traîté dans notre application, le nom de la racine, un pointeur vers la table partielle des noeuds de ce fichier et un pointeur vers la table partielle des places libres. Cette table sera appelée: table des racines.

6. Dans la table des noeuds, le pointeur FAP (father pointer) correspondant à un nom de racine contiendra l'adresse dans la table des racines de l'élément correspondant et le pointeur RTP (root pointer) contiendra l'adresse dans la table des noeuds du dernier noeud crée dans la structure arborescente en suivant le parcours dynastique.

7. Les fichiers à structure arborescente de notre application contiendront des enregistrements de type différent. Les BNB premiers enregistrements seront du type booléen, les autres seront du type caractère .

8. Pour l'analyse organique des traitements, nous avons admis que les fichiers à structure arborescente traités dans notre application étaient au nombre de quatre.

9. Pour faciliter l'analyse organique d'un identifiant de noeud, nous admettrons les conventions suivantes :

   - le nombre de niveaux séparant deux noms de noeud est toujours connu.
   - le nombre de niveaux séparant la racine du premier nom de noeud est soit connu, soit totalement inconnu.

   On exclut donc toute combinaison de symboles * et . ;le symbole * apparaîtra uniquement en tête d'un identifiant de noeud.
   Dans la nouvelle syntaxe d'un identifiant de noeud, la partie ‹séparateur› sera donc :

   ‹séparateur› ::= * | .{.}

10. Nous admettons la possibilité d'écrire un tableau d'actions dont les éléments sont des procédures. (voir TRTGEN)

11. Dans le fichier des commandes FO, les FSKMXL premières lignes contiendront les noms de tous les fichiers utilisés dans notre application. Afin de distingner les fichiers déjà existants de ceux qu'on désire créer, nous écrirons le symbole * devant le nom d'un fichier déjà existant. Ces FSKMXL premières lignes nous permettront de construire une pile FSK des noms de fichier; ces noms étant considérés comme des chaînes de caractère. Nous utiliserons cet artifice car le langage Pascal n'offre pas la possibilité de manipuler des variables 'nom de fichier'. Le symbole * ne sera pas repris dans la pile FSK.

PARAGRAPHE B : DECLARATION.

1. Procedure RD-LIN (var str : stkent)

Comment : cette procédure (read line) lit l'enregistrement physique courant à partir de l'un des fichiers F0, F1,F2,F3,F4.
Elle utilise la procédure de lecture READ.

```
begin
|
|- for idtcrx := 1 to idtmxl do str.idt [idtcrx] := ' ' od;
|
|- case str.fnm of fsk[1] : READ (F0, str.buf);
|                  fsk[2] : READ (F1,str.nxr,str.buf);
|                  fsk[3] : READ (F2,str.nxr,str.buf);
|                  fsk[4] : READ (F3,str.nxr,str.buf);
|                  fsk[5] : READ (F4,str.nxr,str.buf)
|   end;
|
|- idtcrx := 0
|
end
```

2. Procedure GETCAR (var str : stkent)

Comment : cette procédure (get character) met à jour la position courante et fournit le caractère correspondant à cette position.
Elle utilise la procédure RD-LIN.

```
begin
|
|- if idtcrx = idtmxl then RD-LIN (str) fi;
|
|- idtcrx := idtcrx + 1 ;
|
|- crtcar := str.idt[idtcrx]
|
end
```

3. Procedure GETSGNCAR (var str : stkent)

Comment : cette procédure (get significant character) met à jour la position courante et fournit le premier caractère non blanc de l'enregistrement physique courant suivant la position courante.
Elle utilise la procédure GETCAR.

```
begin
|
|- while crtcar = ' ' do GETCAR(str) od ;
|
end
```

4. Procedure PUTCAR (var crtadr : integer)

```
Comment : cette procédure (put character) transfère le caractère courant CRTCAR dans l'enregistrement de sortie REC.
          Elle utilise la procédure WRTREC.

begin
|- if odtcrx = odtmxl then begin
|                          |- WRTREC(crtadr) ;
|                          |- odtcrx := 0
|                          end
|  fi ;
|- odtcrx := odtcrx + 1 ;
|- rec.odt[odtcrx] := crtcar
end
```

5. Procedure WRTREC (var crtadr : integer)

```
Comment : cette procédure (write record) écrit l'enregistrement physique courant dans les fichiers de sortie.
          Elle utilise la procedure SCHFPL qui effectue la recherche d'une place libre dans un fichier et la procédure d'écriture WRITE.

var crtfnm : packed array[1..nnmmxl] of char

begin
|- SCHFPL(crtadr) ;
|- rnb := rnb + 1 ;
|Comment : la variable rnb désigne le nombre d'enregistrements physiques du contenu d'un noeud.
|          Si rnb = 1, il faut mettre à jour le pointeur RCP dans la table des noeuds.
|          Quelque soit la valeur de rnb, l'appel de la procédure WRITE doit être précédé d'une mise à jour du pointeur NXR,de REC,qui contiendra l'adresse de l'enregistrement physique suivant du contenu d'un noeud. Après chaque exécution de la procédure SCHFPL, la variable FPLKPX contiendra l'adresse d'une place libre disponible dans le fichier de sortie.
|
|
|
```

```
- if  rnb = 1  then   begin
                        - ntb[crtadr].rcp := fplkpx ;
                        - crtnxr := fplkpx ;
                        - SCHFPL(crtadr)
                      end
  fi ;
- rec.nxr := fplkpx ;
- crtfnm := ntb[ntb[crtadr].rtp].nnm ;
- case crftnm of fsk[2] : WRITE (F1,crtnxr,rec) ;
                 fsk[3] : WRITE (F2,crtnxr,rec) ;
                 fsk[4] : WRITE (F3,crtnxr,rec) ;
                 fsk[5] : WRITE (F4,crtnxr,rec)
  end ;
- crtnxr := rec.nxr
end
```

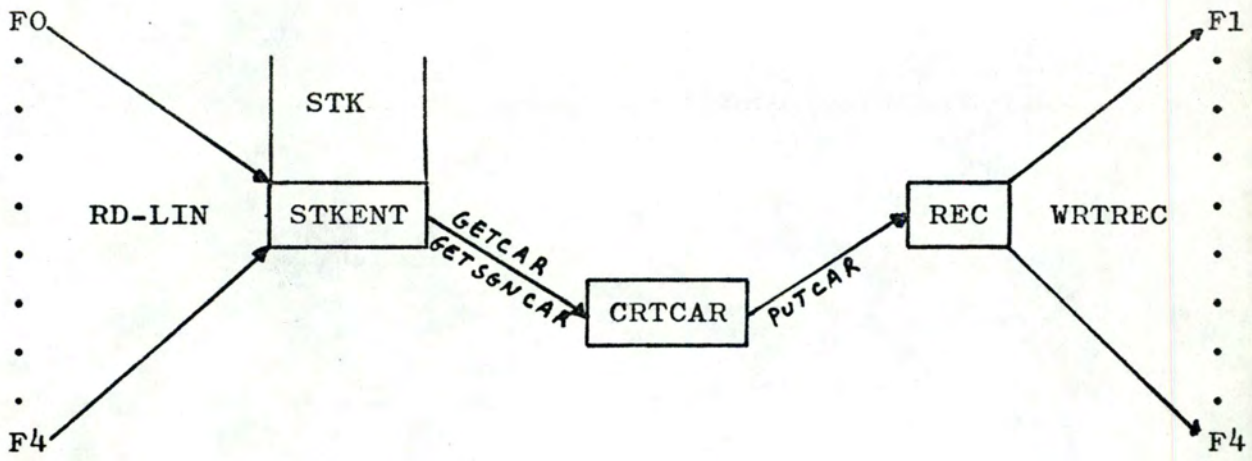
6. Procedure SCHFPL (var crtadr : integer)

Comment : cette procédure (search free place) effectue la recherche d'une place libre dans un fichier de sortie.

```
begin
Comment : les quatres premières instructions permettent de
          retrouver la table partielle des places libres du
          fichier en question; il suffit alors d'effectuer
          un parcours séquentiel de cet élément du tableau
          FTB pour retrouver la première place libre.
          A la sortie de cette procédure la variable FPLKPX
          contient l'adresse logique,dans le fichier de sor-
          tie,de cette place libre.
- ntbkpx := ntb[crtadr].rtp ;
- rtbkpx := ntb[ntbkpx].fap ;
- ftbkpx := rtb[rtbkpx].ftbptr ;
- fplcrx := 1 ;
- while (ftb[ftbkpx][fplcrx] = false) and (fplcrx ≤ fplmxl)
        do  fplcrx := fplcrx + 1  od ;
```

```
|- if  fplcrx > fplmxl  then  message d'erreur : plus de pla-
                                                  ces libres
   fi ;
|- fplkpx := fplcrx ;
|- ftb [ftbkpx] [fplcrx] := false
end
```

7. Schéma



PARAGRAPHE C : INITIALISATION

Procedure TRTINT

```
Comment : cette procédure (treatment initialisation) charge
          en mémoire centrale les tables partielles des
          places libres et les tables partielles des noeuds
          de tous les fichiers à structure arborescente
          traités dans notre application.
          Elle utilise les procédures suivantes :
          READ qui lit l'enregistrement courant à partir
               du fichier d'entrée F0.
          SETFSK qui met à jour la pile FSK des noms de
                 fichier.
          CHGTBS qui effectue le chargement en mémoire cen-
                 trale de la table partielle des places li-
                 bres et de la table partielle des noeuds
                 de chaque fichier.

begin
|- reset F0; reset F1; reset F2; reset F3; reset F4;
|- stkcrx := 1;
|- askcrx := 1;
|- ntbcrx := 0;
|- rtbcrx := 0;
|- ftbcrx := 0;
|- sskcrx := 0;
|- rnb := 0; crtlev := 0;
Comment : on lit successivement les FSKMXL premiers enregis-
|         trements du fichier d'entrée F0 qui contiennent
|         les noms de tous les fichiers à structure arbo-
|         rescente traités dans notre application et on
|         garnit la pile FSK avec les noms de tous ces fi-
|         chiers.
|         Si le nom d'un fichier est précédé du symbole *,
|         on charge en mémoire centrale la table partielle
|         des places libres et la table partielle des noeuds
|         de ce fichier.
|         La variable RTBCRX nous permet de créer la table
|         des racines qui sera construite lors du chargement
|         des tables partielles de tous les fichiers.

|- for fskcrx := 1 to fskmxl
|       do begin
|          |- READ(F0,stk[stkcrx].buf);
|          |- if stk[stkcrx].idt[1] = '*'
|          |       then begin
|          |            |- idtcrx := 1;
|          |            |- SETFSK(fsk[fskcrx]);
|          |            |- CHGTBS
|          |            end
|          |
|          |
```

```
                else begin
                        - idtcrx := 0;
                        - SETFSK(fsk[fskcrx])
                     end
            end
        od;
Comment : les tables partielles des noeuds des différents
          fichiers ayant été chargées, nous pouvons mettre
          à jour le pointeur NTBFRP des places libres dans
          la table des noeuds.
- ntbfrp := ntbcrx + 1;
Comment : il faut chaîner le reste des places libres dans
          la table des noeuds à l'aide du pointeur NXP et
          mettre à blanc les zones NNM restantes.
- for ntbcrx := ntbfrp to ntbmxl
      do begin
          - ntb[ntbcrx].nxp := ntbcrx + 1;
          - ntb[ntbcrx].nnm := ' '
         end
      od;
Comment : il faut mettre à blanc les zones RNM des éléments
          restants de la table des racines.
- rtbkpx := rtbcrx + 1;
- for rtbcrx := rtbkpx to rtbmxl
      do rtb[rtbcrx].rnm := ' ' od;
- INTTRSARG;
Comment : cette procédure (initialisation transition argu-
          ment) met à jour le tableau de transition TRSARG
          de la partie ‹argument› d'une commande GENERATE.
- INTACTARG;
Comment : cette procédure (initialisation action argument)
          met à jour le tableau d'actions associé au ta-
          bleau TRSARG.
- INTTRSNID;
Comment : cette procedure (initialisation transition iden-
          tifier of node) met à jour le tableau de tran-
          sition TRSNID correspondant à un identifiant de
          noeud.
- INTACTNID;
Comment : cette procédure (initialisation action identifier
          of node) met à jour le tableau d'actions associé
          au tableau TRSNID.
- stk[stkcrx].fnm := fsk[1]
Comment : on écrit dans la zone FNM du premier élément de
          la pile STK le nom du fichier d'entrée FO.
end
```

Remarque : pour des raisons de clarté, nous avons préféré écrire directement le contenu des quatres tableaux de transition et d'actions en lieu et place du texte des quatres procédures de mise à jour de ces tableaux.

1. Procedure SETFSK (var crf : packed array 1..nnmmxl of char

```
Comment : cette procédure (set stack of file name) met à
          jour la pile FSK des noms de fichier.

var crfcrx : 0..nnmmxl;

begin
|- crfcrx := 0;
|- while crfcrx ≤ nnmmxl
|         do begin
|            |- crfcrx := crfcrx + 1;
|            |- idtcrx := idtcrx + 1;
|            |- crf[crfcrx] := stk[stkcrx].idt[idtcrx]
|            end
|         od
end
```

2. Procedure CHGTBS

```
Comment : cette procedure (charge tables) effectue le
          chargement en mémoire centrale de la table par-
          tielle des places libres et de la table partielle
          des noeuds de chaque fichier.
          Elle utilise les procedures suivantes :
          CHGFTB qui effectue le chargement de la table
                 partielle des places libres d'un fichier.
          CHGNTB qui effectue le chargement de la table
                 partielle des noeuds d'un fichier.

begin
|- rtbcrx := rtbcrx + 1;
|- ftbcrx := ftbcrx + 1;
|- CHGFTB(fsk[fskcrx]);
|- CHGNTB(fsk[fskcrx])
end
```

2.1. Procedure CHGFTB (var crtfnm : packed array 1..nnmmxl
of char)

```
Comment : cette procedure (charge free table) transfère en
          mémoire centrale la table partielle des places
          libres d'un fichier à structure arborescente.
          Elle utilise la procedure de lecture READ.

var i : integer
```

```
begin
|- i := 0;
|- fplcrx := 0;
|Comment : la constante BNB désigne le nombre d'enregistre-
|          ments physiques du fichier consacré à la table
|          partielle des places libres.
|          Tous ces enregistrements seront placés les uns
|          derrière les autres en mémoire centrale et cons-
|          titueront un élément de la table des places
|          libres.
|- repeat  begin
|          |- i := i + 1 ;
|          |- case crtfnm of fsk[2] : READ(F1,i,bit);
|          |                 fsk[3] : READ(F2,i,bit);
|          |                 fsk[4] : READ(F3,i,bit);
|          |                 fsk[5] : READ(F4,i,bit)
|          |   end ;
|          |- for bitcrx := 1 to bitmxl
|          |       do begin
|          |          |- fplcrx := fplcrx + 1;
|          |          |- ftb[ftbcrx] [fplcrx]  := bit[bitcrx]
|          |          end
|          end
|  until  i = bnb;
|Comment : on met à jour le pointeur FTBPTR de la table
|          des racines.
|- rtb[rtbcrx].ftbptr := ftbcrx
end
```

2.2. Procedure CHGNTB (var crtfnm : packed array [1..nnmmxl] of char)

```
Comment : cette procédure (charge node table) charge en
          mémoire centrale la table partielle des noeuds
          d'un fichier à structure arborescente.
          Elle utilise la procedure SETNTBENT qui remplit
          un élément de la table des noeuds et la procedure
          RD-LIN qui lit l'enregistrement courant à partir
          du fichier d'entrée.

var i, j, k : integer;
    fstidx, lstidx : o..ntbmxl ;
```

```
begin
|
|  Comment : l'adresse logique, dans un fichier à structure
|            arborescente, de l'enregistrement qui contient
|            le début de la table partielle des noeuds est
|            toujours égale à BNB+1.
|            L'enregistrement physique qui contient le début
|            de la table partielle des noeuds doit être traité
|            séparément car il nous permettra de mettre à jour
|            le pointeur NTBPTR de la table des racines et le
|            pointeur FAP (dans la table des noeuds) corres-
|            pondant à un nom de racine. La constante NSE
|            indique le nombre d'éléments de la table des
|            noeuds que peut contenir un enregistrement phy-
|            sique.
|
|  - k := 1; i := 1; j := nnmmx1;
|  - stk[stkcrx].fnm := crtfnm;
|  - stk[stkcrx].nxr := bnb + 1;
|  - RD-LIN(stk[stkcrx]);
|  - while  k ≠ nse + 1
|           do  begin
|               |  - ntbcrx := ntbcrx + 1;
|               |  - SETNTBENT;
|               |  - i := j + 1;
|               |  - j := j + nnmmx1;
|               |  - if  k = 1
|               |    then begin
|               |           |  - rtb[rtbcrx].ntbptr := ntbcrx;
|               |           |  - ntb[ntbcrx].fap := rtbcrx;
|               |           |  - fstidx := ntbcrx
|               |           end
|               |  Comment : la variable FSTIDX (first index)
|               |            mémorise l'adresse logique,dans la
|               |            table des noeuds, du nom de la ra-
|               |            cine.
|               |  - k := k + 1
|               end
|
|  Comment : on lit le reste des enregistrements physiques du
|            contenu d'un noeud et on complète la table des
|            noeuds. Tous ces enregistrements sont chaînés
|            entre eux à l'aide du pointeur NXR. Le dernier
|            enregistrement contient, dans la zone réservée
|            au pointeur NXR, la valeur 0 pour indiquer la fin
|            de la table partielle des noeuds d'un fichier.
|
|
```

```
- repeat  begin
          - RD-LIN(stk[stkcrx]);
          - k := 1 ; i := 1; j := nnmmxl ;
          - while  k ≠ nse + 1
                   do  begin
                       - ntbcrx := ntbcrx + 1;
                       - SETNTBENT;
                       - i := j + 1 ; j := nnmmxl;
                       - if stk[stkcrx].nxr ≠ 0
                           then k := k + 1
                           else k := nse + 1
                         fi
                       end
                    od
          end
  until  stk[stkcrx].nxr = 0

Comment : les tables partielles des noeuds sont placées
          séquentiellement en mémoire centrale dans la
          table des noeuds. Il faut donc remettre à jour
          les pointeurs FAP,RTP et NXP en fonction de
          l'adresse logique,dans la table des noeuds, du
          début de chaque table partielle des noeuds.
          L'adresse du dernier noeud crée dans la table
          sera mémorisée dans la zone réservée au pointeur
          RTP correspondant à un nom de racine.

- ntb[fstidx].rtp := ntbcrx;
- lstidx := ntbcrx;
- for  ntbcrx := fstidx + 1  to  lstidx
       do  ntb[ntbcrx].rtp := fstidx  od ;
- for  ntbcrx := fstidx  to  lstidx - 1
       do  ntb[ntbcrx].nxp := ntbcrx + 1  od ;
- ntb[lstidx].nxp := 0 ;
- for  ntbcrx := fstidx + 1  to  lstidx
       do  ntb[ntbcrx].fap := ntb[ntbcrx].fap + (fstidx - 1
       od ;
end
```

2.2.1. Procedure SETNTBENT

```
Comment : cette procédure (set node table entry) met à
          jour les différentes parties d'un élément de la
          table des noeuds.

begin
|- nnmcrx := 1;
|- for  idtcrx := i  to j
|       do begin
|          |- ntb[ntbcrx].nnm[nnmcrx] :=
|          |                    stk[stkcrx].idt[idtcrx] ;
|          |- nnmcrx := nnmcrx + 1
|          end
|       od ;
|- i := j + 1 ; j := j + 3 ;
|- for idtcrx := i  to j
|     do ntb[ntbcrx].lnb := stk[stkcrx].idt[idtcrx]
|     od ;
|- i := j + 1 ; j := j + 3 ;
|- for idtcrx := i  to j
|     do ntb[ntbcrx].rtp := stk[stkcrx].idt[idtcrx]
|     od ;
|- i := j + 1 ; j := j + 3 ;
|- for idtcrx := i  to j
|     do ntb[ntbcrx].nxp := stk[stkcrx].idt[idtcrx]
|     od ;
|- i := j + 1 ; j := j + 3 ;
|- for idtcrx := i  to j
|     do ntb[ntbcrx].fap := stk[stkcrx].idt[idtcrx]
|     od ;
|- i := j + 1 ; j := j + 3 ;
|- for idtcrx := i  to j
|     do ntb[ntbcrx].rcp := stk[stkcrx].idt[idtcrx]
|     od ;
|
end
```

PARAGRAPHE D : ITERATION

Procedure TRTITE

```
Comment : cette procedure (treatment iteration) effectue
          les instructions successives rencontrées dans le
          fichier d'entrée.
          Elle utilise les procedures suivantes :
          RD-LIN qui lit la ligne courante à partir du
                 fichier d'entrée.
          GETSGNCAR qui met à jour la position courante et
                    fournit le premier caractère non blanc
                    de l'enregistrement physique courant
                    suivant la position courante.
          TRTCRE qui traite un ordre de création.
          TRTEND qui traite un ordre de fin de création
                 d'un noeud.
          TRTINS qui traite un ordre d'insertion.
          TRTDEL qui traite un ordre de suppression.
          TRTADD qui traite un ordre de concaténation de
                 contenus.
          TRTMOD qui traite un ordre de modification.

begin
|
|- while not eof(FO)
|  do begin
|     |
|     |- RD-LIN(stk[stkcrx])
|     |
|     |- GETSGNCAR(stk[stkcrx])
|     |
|     |- if crtcar = '/' then GETSGNCAR(stk[stkcrx])
|     |                  else erreur de syntaxe
|     |  fi ;
|     |
|     |- if (crtcar ≠ 'c') or (crtcar ≠ 'e') or (crtcar ≠ 'i
|     |     or (crtcar ≠ 'd') or (crtcar ≠ 'a') or
|     |     (crtcar ≠ 'm')
|     |  then  erreur de syntaxe
|     |  fi ;
|     |
|     |- case crtcar of c : begin
|     |                     |- GETSGNCAR(stk[stkcrx]);
|     |                     |- TRTCRE
|     |                     end
|     |                  e : begin
|     |                     |- GETSGNCAR(stk[stkcrx]);
|     |                     |- TRTEND
|     |                     end
|     |                  i : begin
|     |                     |-GETSGNCAR(stk[stkcrx]);
|     |                     |- TRTINS
|     |                     end
|     |                  d : begin
|     |                     |- GETSGNCAR(stk[stkcrx]);
|     |                     |- TRTDEL
|     |                     end
|     |
```

```
                        a : begin
                            |-GETSGNCAR(stk[stkcrx]);
                            |- TRTADD
                            end ;
                        m : begin
                            |- GETSGNCAR(stk[stkcrx]);
                            |- TRTMOD
                            end
          end;
        end
end
```

1. Procedure TRTCRE

```
Comment : Cette procedure(treatment create) traite un ordre
          de creation.
          Elle utilise les procedures suivantes:
          GETSGNCAR qui met à jour la position courante et
                    fournit le premier caractère non blanc
                    de l'enregistrement physique courant
                    suivant la position courante.
          TRTSCHLND qui fournit l'adresse logique,dans la
                    table des noeuds,du dernier noeud crée
                    dans un fichier en suivant le parcours
                    dynastique.
          TRTNNM qui traite un nom de noeud.
          TRTCONNODVAL qui traite le contenu d'un noeud par
                       valeur.
          TRTCONNOD qui traite le contenu d'un noeud en lai
                    laissant intactes les commandes GENERA-
                    TE incorporées dans ce contenu.

begin
|Comment : syntaxe de la commande CREATE
|          / C <fonction>,{( nom de fichier )}^1_0 <nom de noeud
|                          {" contenu "}^1_0
|          <fonction> ::= V|R
|- rnb := 0;
|- if crtcar = 'v' then begin
|                       |- vrf-sw := true;
|                       |- GETSGNCAR(stk[stkcrx])
|                       end
|                  else if crtcar = 't'
|                       then begin
|                            |- vrf-sw := false;
|                            |- GETSGNCAR(stk[stkcrx])
|                            end
|                       else erreur de syntaxe
|                       fi
|  fi;
|- if crtcar = ',' then GETSGNCAR(stk[stkcrx])
|                  else erreur de syntaxe
|  fi;
```

```
|
|- if crtcar = '(' then begin
|                         |- TRTSCHLND;
|                         |- GETSGNCAR(stk[stkcrx])
|                        end
|  fi;
|
|- TRTNNM;
|
|- if crtcar ≠ eoln then
|                    if vrf-sw = true then TRTCONNODVAL(lnd)
|                                     else TRTCONNOD(lnd)
|                    fi
|  fi
end
```

1.1. Procedure TRTSCHLND

Comment : cette procédure (treatment search last node) est est appelée dans le cas où on désire reprendre la création d'un fichier à structure arborescente. Elle permet de retruver l'adresse logique,dans la table des noeuds,du dernier noeud crée dans le fichier en suivant le parcours dynastique. Elle utilisela procedure CETSGNCAR qui met à jour la position courante et fournit le premier caractère non blanc de 1'enregistrement physique courant suivant la position courante.

```
var crf : packed array [1..NNMMXL] of char;
    crfcrx : 0..NNMMXL;

begin
|
|Comment : on affecte la chaîne de caractères ‹ nom de fichier› à la variable locale current file (crf).
|
|- for crfcrx := 1 to nnmmxl do crf[crfcrx] := ' ' od;
|
|- crfcrx := 0;
|
|- while crtcar ≠ ')' do begin
|                         |- GETSGNCAR (stk[stkcrx]);
|                         |- crfcrx := crfcrx + 1;
|                         |- crf[crfcrx] := crtcar
|                        end;
|
|Comment : on effectue un parcours séquentiel de la table des racines pour vérifier si le fichier en question a déjà été crée. Si le fichier n'existe pas, on envoie un message d'erreur. Si le fichier existe, on utilise les pointeurs NTBPTR de la table des racines et RTP de la table des noeuds pour retrouver l'adresse du dernier noeud crée et la mémoriser dans la variable LND.
|
|- rtbcrx := 1;
|
```

```
   - while (rtb[rtbcrx].rnm ≠ crf) and (rtbcrx ≤ rtbmx1)
           do rtbcrx := rtbcrx + 1 od;

   - if rtbcrx > rtbmx1 then message d'erreur, le fichier
                             n'a pas été crée
                        else begin
                             |- ntbkpx := rtb[rtbcrx].ntbptr;
                             |- lnd := ntb[ntbkpx].rtp
                             end
     fi
end
```

1.2. Procedure TRTNNM

```
Comment : cette procédure (treatment name of node) traite
          un nom de noeud.
          Elle utilise les procédures :
          TRTROT qui traite le noeud 'racine d'une structu-
                 re arborescente'.
          TRTNOD qui traite un noeud quelconque.
          SETCRN qui affecte les caractères courants corres-
                 pondant à < nom de noeud > à la variable
                 current name (CRN).

begin
|- SETCRN;
|
|- if crtlev = 0 then TRTROT
|                else TRTNOD
|  fi
end
```

1.2.1. Procedure SETCRN

```
Comment : cette procedure (set current name) affecte les
          caractères courants correspondant à < nom de noeud >
          à la variable CRN.
          Elle utilise la procédure GETSGNCAR qui met à
          jour la position courante et fournit le premier
          caractère non blanc de l'enregistrement physique
          courant suivant la position courante.

begin
|- for crncrx := 1 to nnmmx1 do crn[crncrx] := ' ' od;
|
|- crncrx := 0;
|
|- while (crtcar ≠ '¤') or (crtcar ≠ eoln)
|        do begin
|           |- crncrx := crncrx + 1;
|           |- crn[crncrx] := crtcar;
|           |- GETSGNCAR(stk[stkcrx])
|           end
end
```

1.2.2. Procedure TRTROT

```
Comment : cette procédure traite le noeud qui est la racine
          d'une structure arborescente.
          Elle utilise la procédure TSTROTCMP qui recherche
          une place libre dans la table des racines.

begin
|
|- TSTROTCMP;
|
|Comment : mise à jour de la table des racines.
|- rtb[rtbkpx].rnm := crn;
|- rtb[rtbkpx].ntbptr := ntbfrp;
|Comment : mise à jour de la table des noeuds, de la varia-
|          ble LND, du pointeur des places libres dans la
|          table des noeuds (NTBFRP), du numéro de niveau
|          LNB, du pointeur NXP pour indiquer la fin de la
|          table partielle des noeuds. Signalons que le poin-
|          teur FTBPTR a déjà été mis à jour dans le para-
|          graphe Initialisation.
|
|- ntbcrx := ntbfrp;
|- ntb[ntbcrx].nnm := crn;
|- ntb[ntbcrx].lnb := crtlev;
|- ntb[ntbcrx].fap := rtbkpx;
|- ntb[ntbcrx].rtp := ntbcrx;
|- lnd := ntbcrx;
|- ntbfrp := ntb[ntbcrx].nxp;
|- ntb[ntbcrx].nxp := 0;
|- crtlev := crtlev + 1
end
```

i. Procedure TSTROTCMP

```
Comment : cette procedure (test root compatible) recherche
          la première place libre dans la table des racines.

begin
|- rtbcrx := 0;
|Comment : on parcourt séquetiellement la table des racines.
|          La variable RTBKPX mémorise l'adresse logique
|          dans la table des racines de la première place
|          libre.
|- repeat  rtbcrx := rtbcrx + 1
|  until  (rtb[rtbcrx].rnm ≠ crn) or (rtb[rtbcrx].rnm = ' ')
|         or (rtbcrx = rtbmxl);
|
|
|
```

```
- if rtb[rtbcrx].rnm = ' '
     then  rtbkpx := rtbcrx
     else  begin
             - if rtbcrx = rtbmxl
                  then message d'erreur, plus de places
                       libres dans la table des racines
               fi;
             - if rtb[rtbcrx].rnm = crn
                  then message d'erreur, le fichier a déjà
                       été créé
               fi
           end
  fi
end
```

1.2.3. Procedure TRTNOD

```
Comment : cette procédure (treatment node) traite un noeud
          quelconque.
          Elle utilise la procédure TSTNODCMP qui détecte
          l'existence éventuelle dans la table des noeuds
          d'un noeud dont le chemin qui le relie à la ra-
          cine serait identique à celui du noeud qu'on
          désire créer.

var oldlev : integer;
    ntbidx : 0..ntbmxl;

begin
- TSTNODCMP;
Comment : le pointeur NTBFRP indique la place libre dis-
          ponible dans la table des noeuds; on effectue la
          mise à jour des éléments NNM et LNB correspon-
          dant à cette place libre.
- ntbcrx := ntbfrp;
- ntb[ntbcrx].nnm := crn;
- ntb[ntbcrx].lnb := crtlev
Comment : mise à jour des pointeurs FAP et RTP.
          La variable CRTLEV contient le numéro de niveau
          du noeud qu'on désire créer.
          Pour mettre à jour le pointeur père de ce noeud
          (FAP), il faut retrouver son frère ainé dans la
          table des noeuds (même numéro de niveau). Pour ce
          faire,on remonte dans la table des noeuds avec
          les pointeurs FAP à partir du dernieur noeud
          crée (LND).
- ntbidx := lnd;
- oldlev := crtlev;
```

```
- while  ntb[ntbidx].lnb > oldlev
         do  ntbidx := ntb[ntbidx].fap  od;
- ntb[ntbcrx].fap := ntb[ntbidx].fap;
- ntb[ntbcrx].rtp := ntb[ntbidx].rtp;
Comment : mise à jour de la variable LND et mémorisation
          du contenu de cette variable dans la zone réser-
          vée au pointeur FAP correspondant au nom de la
          racine de la structure arborescente.
- lnd := ntbfrp;
- ntb[ntb[ntbcrx].rtp].fap := lnd;
Comment : mise à jour du pointeur NTBFRP; la zone réservée
          au pointeur NXP contient la valeur 0 pour indi-
          quer la fin de la table partielle des noeuds.
          Mise à jour du numéro de niveau courant.
- ntbfrp := ntb[ntbcrx].nxp;
- ntb[lnd].nxp := 0;
- crtlev := crtlev + 1
end
```

i. Procedure TSTNOCCMP

```
Comment : cette procedure (test node compatibility) recher-
          che dans la table des noeuds l'existence éven-
          tuelle d'un noeud dont le chemin qui le relie à la
          racine serait identique à celui du noeud qu'on
          désire créer.

begin
Comment : la variable CRTLEV contient le numéro de niveau
          du noeud qu'on désire créer. On recherche le père
          de ce noeud (première instruction while); on
          effectue ensuite le parcours dynastique du sous-
          arbre de ce père et on vérifie s'il existe sur ce
          parcours un noeud de même nom et de même numéro
          de niveau que le noeud qu'on désire créer.
          (deuxième instruction while)

- ntbidx := lnd;
- oldlev := crtlev;
- while  ntb[ntbidx].lnb ≠ crtlev - 1
         do  ntbidx := ntb[ntbidx].fap  od;
- oldlev := ntb[ntbidx].lnb;
- ntbidx := ntb[ntbidx].nxp;
```

```
- while  ntb[ntbidx].lnb > oldlev
         do  begin
             - if  (ntb[ntbidx].lnb = crtlev + 1)
                   and (ntb[ntbidx].nnm=crn )

               then message d'erreur, il existe déjà un
                    noeud de même nom et de même numéro de
                    niveau

               else ntbidx := ntb[ntbidx].nxp
               fi
             end
         od
end
```

1.3. Procedure TRTCONNOD (var crtadr : integer)

```
Comment : cette procédure (treatment content of node) traite
          le contenu d'un noeud en laissant intactes les
          commandes GENERATE incorporées dans le texte du
          contenu.
          Elle utilise les procédures suivantes :
          GETCAR qui met à jour la position courante et
                 fournit le caractère correspondant à cette
                 position.
          PUTCAR qui transfère le caractère courant dans
                 l'enregistrement de sortie.
          TRTENDCRE qui traite la fin de création du con-
                    tenu d'un noeud.

var abscar : char;

begin
Comment : cette procédure permet d'insérer des " à l'inté-
          rieur d'un contenu à condition de les dédoubler.

- if crtcar = '"' then erreur de syntaxe fi;
- repeat  begin
            - GETCAR(stk[stkcrx] );
            - if crtcar = '"' then begin
                                     - GETCAR(stk[stkcrx] );
                                     - if crtcar = '"'
                                       then crtcar := etx
                                       fi
                                   end
              fi;
            - abscar := crtcar;
            - PUTCAR(crtadr)
          end
```

```
    until abscar = etx;
  - TRTENDCRE(crtadr)
end
```

1.4. Procedure TRTCONNODVAL (var crtadr : integer)

```
Comment : cette procedure traite le contenu d'un noeud par
          valeur.
          Elle utilise les procedures suivantes :
          GETCAR qui met à jour la position courante et
                 fournit le caractère correspondant à cette
                 position.
          GETSGNCAR qui met à jour la position courante et
                    fournit le premier caractère non blanc
                    de l'enregistrement courant suivant la
                    position courante.
          PUTCAR qui transfère le caractère courant dans
                 l'enregistrement de sortie.
          TRTENDCRE qui traite la fin de création du conte-
                    nu d'un noeud.
          TRTGEN qui traite une commande GENERATE.

var abscar : char;

begin
Comment : cette procedure (treatment content node by value)
          permet d'insérer des " à l'intérieur d'un con-
          tenu à condition de les dédoubler.
          La variable ASKCRX est initialisée à la valeur 1
          au premier appel de la procedure TRTGEN; la pile
          ASK contient les adresses logiques dans la table
          des noeuds de tous les noeuds identifiés par
          l'identifiant de noeud de la commande de générati

- if crtcar ≠ '"' then erreur de syntaxe fi;
- repeat  begin
            - GETCAR(stk[stkcrx]);
            - if  crtcar = '/'
                  then begin
                         - GETSGNCAR(stk[stkcrx]);
                         - if  crtcar = 'g'
                               then begin
                                      - GETSGNCAR(stk[stkcrx])
                                      - TRTGEN(crtadr);
                                      - GETCAR(stk[stkcrx])
                                    end
                               else erreur de syntaxe
                           fi
                       end
              fi;
```

```
            - if crtcar = '„' then begin
                                    - GETCAR(stk [stkcrx]);
                                    - if crtcar = '„'
                                      then crtcar := etx
                                      fi
                                    end
              fi;
            - abscar := crtcar;
            - PUTCAR(crtadr)
          end
    until abscar = etx;
  - TRTENDCRE(crtadr)
end
```

1.5. Procedure TRTENDCRE (var crtadr : integer)

Comment : cette procédure (treatement end of create) traite la fin de création du contenu d'un noeud.
Elle utilise la procédure SCHFPL qui effectue la recherche d'une place libre dans un fichier de sortie.

```
var crtfnm : packed array [1..nnmmxl] of char

begin
- rnb := rnb + 1;

Comment : la variable RNB désigne le nombre d'enregistrements physiques du contenu d'un noeud.
          Si RNB = 1, il faut rechercher une place libre dans le fichier (procédure SCHFPL), mettre à jour le pointeur RCP dans la table des noeuds et mémoriser l'adresse de cette place libre dans la variable CRTNXR avant l'appel du WRITE.
          Dans le cas contraire, il suffit d'appeler la procédure WRITE puisque la variable CRTNXR a déjà été mise à jour dans la procédure WRTREC.
          La zone de REC réservée au pointeur NXR contient la valeur 0 pour indiquer qu'il s'agit du dernier enregistrement physique du contenu d'un noeud.

- if rnb = 1 then begin
                  - SCHFPL(crtadr);
                  - ntb[crtadr].rcp := fplkpx;
                  - crtnxr := fplkpx
                  end
  fi;
```

```
|- rec.nxr := 0;
|- crtfnm := ntb[ntb[crtadr].rtp].nnm ;
|- case crtfnm of fsk[2] : WRITE(F1,crtnxr,rec) ;
                  fsk[3] : WRITE(F2,crtnnx,rec) ;
                  fsk[4] : WRITE(F3,crtnxr,rec) ;
                  fsk[5] : WRITE(F4,crtnxr,rec)
  end ;
|- rnb := 0
end
```

---

2. Procedure TRTGEN

2.1. Introduction

Rappelons la syntaxe de la commande GENERATE :

/ G ‹argument› ($\{!\}_0^1$ ‹identifiant de noeud›)

‹argument› ::= ‹empty› | ‹ω› | ‹ω›‹entier› | ‹ω›,‹entier› | ω‹entier›,‹entier›

‹identifiant de noeud› ::= $\{$‹nom de noeud›$\}_0^1$ ‹queue d'identifiant›

‹queue d'identifiant› ::= ‹séparateur›‹nom de noeud› | ‹queue d'identifiant›‹séparateur›‹nom de noeud›

‹séparateur› ::= * | .{.} (suite à nos conventions)

‹nom de noeud› ::= ‹lettre› {‹lettre ou chiffre›}

‹lettre ou chiffre› ::= ‹lettre› | ‹chiffre›

Nous allons diviser le traitement de la commande GENERATE en deux parties :
- le traitement de la partie ‹argument› : procedure TRTARG
- le traitement de la partie ‹identifiant de noeud› : procedure TRTNID

Pour chacun de ces traitements, nous utiliserons le principe des tables de transition.
L'algorithme de la procedure TRTARG pourra être représenté par un diagramme de transition, lui-même représenté par un tableau du type : transition [0..nombre d'états,1..ncar] ; ncar désigne le nombre de caractères possibles de ‹argument›. A chacun de ces caractères sera associé un code indiquant son type (,digit,...). Nombre d'états désigne le nombre d'états du tableau autre que l'état 0.
Ainsi, si état est un numéro d'état et code(crtcar) est le code associé au caractère lu crtcar,
transition [état,code(crtcar)] représente le numéro de l'état auquel on arrive à partir de l'état : état.
A ce tableau de transition sera associé un tableau de procédures, de même dimension, qui indiquera la procedure à effectuer pour chaque élément du tableau de transition.

Tableaux de transition et tableaux d'actions correspondant aux procédures TRTARG et TRTNID.

TRSARG

code car / états	1 ω	2 ,	3 (	4 dgt	5 ≠
0	1	-2	-1	-2	-2
1	-2	3	-1	2	-2
2	-2	3	-1	-2	-2
3	-2	-2	-2	4	-2
4	-2	-2	-1	-2	-2

TRSNID

code car / états	1 !	2 *	3 .	4 let dgt	5)	6 ≠
0	1	2	3	4	-2	-2
1	-2	2	3	4	-2	-2
2	-2	-2	-2	4	-2	-2
3	-2	-2	3	4	-2	-2
4	-2	-2	3	4	-1	-2

ACTARG

code car. / états	1 ω	2 ,	3 (	4 dgt	5 ≠
0	-	ERR	A03	ERR	ERR
1	ERR	-	A13	A14	ERR
2	ERR	-	A23	ERR	ERR
3	ERR	ERR	ERR	A34	ERR
4	ERR	ERR	-	ERR	ERR

ACTNID

code car. / états	1 !	2 *	3 .	4 let dgt	5)	6 ≠
0	A01	-	A03	A04	ERR	ERR
1	ERR	-	A13	A14	ERR	ERR
2	ERR	ERR	ERR	A24	ERR	ERR
3	ERR	ERR	A33	A34	ERR	ERR
4	ERR	ERR	A43	A44	A45	ERR

- : procedure NIL
ERR: erreur de syntaxe
Act 03 : procedure ABSARG
Act 13 : procedure ABSFSTSECNBR
Act 23 : procedure ABSSECNBR
Act 14 : procedure TRTFSTNBR
Act 34 : procedure TRTSECNBR

ERR : traitement d'erreur
- : procedure NIL
Act 01 : procedure TRTONE
Act 03 : procedure LEVDST
Act 13 : procedure LEVDST
Act 33 : procedure LEVDST
Act 43 : procedure TRSNNMLEV
Act 04 : procedure INTTRTNNM
Act 14 : procedure INTTRTNNM
Act 24 : procedure TRTSTA
Act 34 : procedure INTTRTNNM
Act 44 : procedure ITETRTNNM
Act 45 : procedure SCHNDS

2.2. Procedure TRTGEN (var crtadr : integer)

```
Comment : cette procédure (treatment GENERATE) traite un
          ordre de création.
          Elle utilise les procédures suivantes :
          TRTARG qui traite l'argument d'une commande de
                 génération.
          TRTNID qui traite un identifiant de noeud.
          GEN qui génére les contenus de tous les noeuds
              identifiés par ‹identifiant de noeud›.

const sskmxl ;
type sskent  record ntbcrx:0..ntbmxl;
                    askkpx:0..askmxl
             end
var fstnbr, secnbr : integer ;
    ssk : array [1..sskmxl] of sskent ;
    sskcrx : 0..sskmxl ;
Comment : dans la pile SSK, on sauve avant chaque appel
          récursif de la procedure TRTGEN la valeur de la
          variable NTBCRX correspondant au noeud dont on
          est occupé à générer le contenu et la valeur de
          la variable ASKKPX qui indique la hauteur dans
          la pile ASK du premier noeud de la famille de
          noeuds désignés par un identifiant de noeud.
          Les variables FSTNBR etSECNBR seront utilisées
          dans la procédure TRTARG

begin
| - TRTARG;
| - TRTNID;
| - if (mny = false) and (nnb > 1)
|       then message d'erreur,‹identifiant de noeud›désigne
|            une famille de noeuds.
|       else  if nnb = 0
|                  then message d'erreur, il n'y a aucun noeud
|                       identifié par ‹ identifiant de noeud ›.
|                  else GEN(crtadr)
|             fi
|   fi
end
```

2.2.1. Procedure TRTARG

```
Comment : cette procédure (treatment argument) traite la
          partie < argument > d'une commande GENERATE.
          Elle utilise les fonctions :
          CODARG qui fournit le code du caractère courant
                 d'un argument.
          VALNBR qui fournit la valeur de l'élément < entier >
                 de < argument >
          et les procédures :
          ABSARG qui traite le cas où < argument > = empty.
          ABSFSTSECNBR qui traite le cas où < argument > = ω
          ABSSECNBR qui traite le cas où < argument > =
                    ω < entier >
          TRTFSTNBR qui met à jour la variable FSTNBR.
          TRTSECNBR qui met à jour la variable SECNBR.
          GETSGNCAR qui met à jour la position courante
                    et fournit le premier caractère non
                    blanc de l'enregistrement physique
                    courant suivant la position courante.

var ste, oldste : integer;

begin
|Comment : la variable STE(état) est initialisée à 0 comme
|          point de départ du diagramme de transition.
|          A la fin de l'exécution de cette procédure, les
|          variables FSTNBR et SECNBR contiendront les
|          valeurs des numéros de niveau spécifiés dans
|          <argument>.
|
|- ste := 0;
|- while ste ≥ 0
|        do  begin
|            |- oldste := ste ;
|            |- ste := trsarg[oldste,CODARG] ;
|            |- actarg [oldste,CODARG ] ;
|            |- GETSGNCAR(stk[stkcrx] )
|            end
|        od
end
```

i. Function CODARG : integer

```
Comment : cette fonction (code argument) fournit le code
          du caractère courant de < argument >

var dgt : set of '0'..'9' ;
```

```
begin
|- if (crtcar ≠ ω) or (crtcar ≠ ',') or (crtcar ≠ '(') or
|     (crtcar not in dgt)
|     then CODARG := 5
|  fi ;
|- case crtcar of ω : CODARG := 1 ;
|                 , : CODARG := 2 ;
|                 ( : CODARG := 3 ;
|               dgt : CODARG := 4
|  end
end
```

ii. Function VALNBR : char

Comment : cette fonction (value number) fournit la valeur de l'élément ‹ entier › de ‹ argument › .
Elle utilise la procédure GETSGNCAR qui met à jour la position courante et fournit le premier caractère non blanc de l'enregistrement physique courant suivant la position courante.

```
var oldcrtcar : char;
    dgt : set of '0'..'9' ;
begin
|- oldcrtcar := crtcar ;
|- GETSGNCAR(stk[stkcrx]) ;
|- while (crtcar in dgt)
|        do  begin
|             |- oldcrtcar := oldcrtcar * 10 + crtcar ;
|             |- GETSGNCAR(stk[stkcrx])
|             end ;
|- VALNBR := oldcrtcar
end
```

iii. Procedure ABSARG

Comment : cette procédure (absence of argument) traite le cas où ‹ argument › = empty.

```
begin
|- fstnbr := 0 ;
|- secnbr := 0
end
```

iv. Procedure ABSFSTSECNBR

Comment : cette procédure (absence of first and second number) traite le cas où ⟨argument⟩ = ω .

```
begin
|- fstnbr := 0 ;
|- secnbr := maxint
end
```

v. Procedure ABSSECNBR

Comment : cette procédure (absence of second number) traite le cas où ⟨argument⟩ = ω ⟨entier⟩ .

```
begin
|- secnbr := maxint
end
```

vi. Procedure TRTFSTNBR

Comment : cette procédure (treatment of first number) met à jour la variable FSTNBR.
Elle utilise la fonstion VALNBR qui fournit la valeur d'un élément ⟨entier⟩de ⟨argument⟩.

```
begin
|- fstnbr := VALNBR
end
```

vii. Procedure TRTSECNBR

Comment : cette procédure (treatment of second number) met à jour la variable SECNBR.
Elle utilise la fonction VALNBR qui fournit la valeur d'un élément ⟨entier⟩ de ⟨argument⟩ .

```
begin
|- secnbr := VALNBR
end
```

2.2.2. Procedure TRTNID

Comment : cette procedure (treatment identifier of node) analyse un identifiant de noeud.
Elle utilise la fonction CODNID qui fournit le code du caractère courant d'un identifiant de noeud et les procedures suivantes :
GETSGNCAR qui met à jour la position courante et fournit le premier caractère non blanc de l'enregistrement physique courant suivant la position courante.
TRTONE qui traite le cas où crtcar = !
LEVDST qui calcule le nombre de niveaux séparant deux noms de noeud.
TRSNNMLEV qui traite le passage de l'état crtcar = letter or digit à l'état crtcar = '.'
INTTRTNNM qui initialise le traitement d'un nom de noeud et traite la fin du calcul d'un numéro de niveaux.
TRTSTA qui traite le cas où crtcar = *.
ITETRTNNM qui continue le traitement d'un nom de noeud.
SCHNDS qui recherche les adresses logiques dans la table des noeuds, du noeud ou de la famille de noeuds désignés par un identifiant.

```
const nskmxl ;
type name : array[1..nnmmxl] of char ;
var ste, oldste : integer ;
    nsk : array[1..cmnmxl] of name ;
    nme : name ;
    nmecrx : 0..nnmmxl ;
    lsk : array[1..cmnmxl] of integer ;
    cmncrx : 0..cmnmxl ;
    cmnkpx : 0..cmnmxl ;
    levnbr : integer ;
```

Comment : considérons l'identifiant de noeud ' $d_{01} n_1 d_{12} n_2 \ldots n_{k-1} d_{k-1,k} n_k$ ' où d_{ij} représente le nombre de niveaux séparant le noeud de nom n_i du noeud de nom n_j .
Pendant l'exécution de la procédure TRTNID et avant l'appel de la procédure SCHNDS, les différents noms n_i seront enregistrés dans une pile des noms de noeud (NSK) et les valeurs d_{ij} dans une pile des nombres de niveaux (LSK).
Ces deux piles auront toujours la même hauteur au cours de l'exécution de la procédure TRTNID; un seul indice de pile sera donc suffisant CMNCRX (common current index).
La constante CMNMXL indique la hauteur maximale de ces deux piles.
La variable CMNKPX permet de mémoriser la hauteur dans la pile, correspondant au noeud de nom n_k .
La pile des adresses ASK contiendra les adresses logiques dans la table des noeuds NTB de tous les noeuds identifiés par un identifiant de noeud.
Cette table sera mise à jour pendant l'exécution de la procédure SCHNDS.

```
begin
|- levnbr := 0 ;
|- cmncrx := 1 ;
|- nmecrx := 1 ;
|- nnb := 0 ;
|- sskcrx := sskcrx + 1 ;
|- mny := true ;
|- ste := 0 ;
|Comment : la variable état est initialisée à 0 comme point
|          de départ du diagramme de transition.
|          On met à blanc le contenu de la pile NSK.
|
|- for cmncrx := 1 to cmnmx1
|      do for nmecrx := 1 to nnmmx1
|         do nsk[cmncrx] [nmecrx] := ' '
|         od
|      od ;
|- while ste ≥ 0 do begin
|                   |- oldste := ste ;
|                   |- ste := trsnid[oldste,CODNID ] ;
|                   |- actnid[oldste,CODNID ] ;
|                   |- GETSGNCAR(stk[stkcrx] )
|                   end
end
```

i. Function CODNID : integer

```
Comment : cette fonction (code identifier of node) fournit
          le code du caractère courant de l'identifiant de
          noeud.

var letdgt : set of ['0'..'9','a'..'z' ] ;
begin
|- if (crtcar ≠ '!') or (crtcar ≠ '*') or (crtcar ≠ '.') or
|     (crtcar not in letdgt) or (crtcar ≠ ')')
|  then CODNID := 6  fi ;
|- case crtcar of ! : CODNID := 1 ;
|                 * : CODNID := 2 ;
|                 . : CODNID := 3 ;
|            letdgt : CODNID := 4 ;
|                 ) : CODNID := 5
|  end
end
```

ii. Procedure TRTONE

Comment : cette procédure (treatment one) traite le cas où crtcar = '!'.

```
begin
|- mny := false
end
```

iii. Procedure LEVDST

Comment : cette procédure (level distance) calcule le nombre de niveaux séparant deux noms de noeud.

```
begin
|- levnbr := levnbr + 1
end
```

iv. Procedure TRSNNMLEV

Comment:cette procédure (transition node name level) traite le passage de l'état crtcar = letter or digit à l'état crtcar = '.'

```
begin
|- cmncrx := cmncrx + 1 ;
|- levnbr := levnbr + 1
end
```

v. Procedure INTTRTNNM

Comment : cette procédure (initialisation treatment of node name) initialise le traitement d'un nom de noeud et traite la fin du calcul d'un nombre de niveaux.

```
begin
|- lsk[cmncrx] := levnbr ;
|- nsk[cmncrx][nmecrx] := crtcar
end
```

vi. Procedure TRTSTA

```
Comment : cette procedure (treatment star) traite le cas
          où crtcar = 'x'

begin
|Comment : pour indiquer que le nombre de niveaux est
|          inconnu, on utilise la valeur - 1
|- lsk[cmncrx] := - 1 ;
|- nsk[cmncrx][nmecrx] := crtcar
end
```

vii. Procedure ITETRTNNM

```
Comment : cette procédure (iteration treatment of node name)
          continue le traitement d'un nom de noeud.

begin
|- nmecrx := nmecrx + 1 ;
|- nsk[cmncrx][nmecrx] := crtcar
end
```

viii. Procedure SCHNDS

```
Comment : cette procédure (search nodes) recherche les
          adresses logiques dans la table des noeuds de
          tous les noeuds identifiés par un identifiant
          de noeud.
          Elle utilise la procédure ANAWAY qui recherche
          l'existence éventuelle dans la table des noeuds
          d'un noeud dont le chemin qui le relie à la raci-
          ne peut être décrit par < identifiant de noeud>.

begin
|Comment : on effectue un parcours séquentiel pour retrou-
|          ver l'adresse logique (NTBKPX) dans la table des
|          noeuds (NTB) du dernier noeud mentionné dans
|          <identifiant de noeud>. Le nom de ce noeud se
|          trouve au sommet de la pile des noms de noeud.
|          (NSK). La variable CMNKPX mémorise la hauteur de
|          la pile NSK.
|- ntbcrx := 1 ;
|- cmnkpx := cmncrx ;
|- while ntbcrx ≤ ntbmxl
|        do  begin
|             |- if ntb[ntbcrx].nnm ≠ nsk[cmnkpx]
|             |      then ntbcrx := ntbcrx + 1
```

```
                else  begin
                        - ntbkpx := ntbcrx ;
                        - ANAWAY
                      end
            fi
        end
end
```

ix. Procedure ANAWAY

Comment : cette procédure (analyse way) recherche l'existence éventuelle dans la table des noeuds d'un noeud dont le chemin qui le relie à la racine peut être décrit par l'identifiant de noeud. Elle utilise la procédure UPDASK qui mémorise, dans la pile des adresses ASK, l'adresse logique dans la table des noeuds, d'un noeud de la famille de noeuds désignés par identifiant de noeud.

```
var idnnnm : boolean ;
        i : integer ;

begin
Comment : l'algorithme de cette procédure a déjà été décrit
          page 27 pour résoudre le problème : à partir
          d'un identifiant de noeud d01 n1 d12 n2...nk , re-
          trouver le noeud ou la famille de noeuds dési-
          gnés par cet identifiant".
          La variable IDNNNM (identical node name) aura la
          valeur true lorsqu'il y aura correspondance entre
          le nom du noeud trouvé dans la table des noeuds
          NTB et celui mentionné dans la pile NSK.
          L'algorithme se termine lorsque CMNCRX = 1 et
          que LSK[1] = -1 (cas où d01 = *) ou LSK[1] = d01
          (d01 est égal au numéro de niveau du noeud de
          nom n1.
- idnnnm := true ;
- while (cmncrx>1) and (idnnnm = true)
        do  begin
            - for i := 1  to  lsk[cmncrx]
                 do  ntbcrx := ntb[ntbcrx].fap  od ;
            - cmncrx := cmncrx - 1 ;
            - if ntb[ntbcrx].nnm = nsk[cmncrx]
                 then idnnnm := false
              fi ;
```

```
|                |- if idnnnm = true
|                |      then if (lsk[cmncrx] = -1) or
|                |                (lsk[cmncrx] = ntb[ntbcrx].lnb)
|                |                then UPDASK
|                |           fi
|                |   fi
|                end
|        od ;
|- ntbcrx := ntbkpx + 1 ;
|- cmncrx := cmnkpx
end
```

x. Procedure UPDASK

```
Comment : cette procédure ( update stack of address) mémo-
          rise dans la pile des adresses ASK, l'adresse
          logique dans la table des noeuds NTB, d'un noeud
          de la famille de noeuds désignés par l'identifiant
          de noeud; elle incrémente le compteur NNB du
          nombre de noeuds identifiés.

begin
|Comment : la variable ASKKPX sert à mémoriser la valeur
|          de ASKCRX correspondant au premier noeud de la
|          famille de noeuds identifiés par l'identifiant
|          de noeud.
|- nnb := nnb + 1 ;
|- ask[askcrx] := ntbkpx ;
|- if nnb = 1 then askkpx := askcrx fi ;
|- askcrx := askcrx + 1
end
```

2.2.3. Procedure GEN (var crtadr : integer)

```
Comment : cette procédure (generate) génère les contenus de
          tous les noeuds identifiés par < identifiant de
          noeud > .
          Elle utilise la procédure GENREC qui génère le
          contenu d'un des noeuds identifiés par < identi-
          fiant de noeud > .

var oldlev : integer ;
```

```
begin
  Comment : la pile des adresses ASK fournit les adresses
            logiques dans la table des noeuds, de tous les
            noeuds désignés par < identifiant de noeud > .
            Pour le sous-arbre de chacun de ces noeuds, il
            faut rechercher tous les noeuds, dont le numéro
            de niveau est compris entre la valeur de FSTNBR
            et la valeur de SECNBR ; ces deux variables
            ayant été mises à jour pendant l'exécution de
            la procédure TRTARG.
  - repeat  begin
              - askcrx := askcrx - 1 ;
              - ntbcrx := ask[askcrx] ;
              - oldlev := ntb[ntbcrx].lnb ;
              - if (fstnbr = 0) and secnbr = 0)
                  then GENREC(crtadr)
                  else repeat
                         begin
                           - if ntb[ntbcrx].lnb ≤ oldlev +
                                                   secnbr
                             then if ntb[ntbcrx].lnb ≥
                                              oldlev + fstnbr
                                  then GENREC(crtadr)
                                  fi
                             fi
                           - ntbcrx := ntb[ntbcrx].nxp
                         end
                       until ntb[ntbcrx].lnb = oldlev
                fi
            end
    until askcrx := askkpx
end
```

i. Procedure GENREC (var crtadr : integer)

Comment : cette procédure (generate record) génère le contenu d'un noeud appartenant à la famille des noeuds identifiés par identifiant de noeud .
Elle utilise les procédures suivantes :
RD-LIN qui lit l'enregistrement physique courant à partir du fichier d'entrée.
GETCAR qui met à jour la position courante et fournit le caractère correspondant à cette position.
TRTGEN qui traite un ordre de génération.
PUTCAR qui transfère le caractère courant CRTCAR dans l'enregistrement de sortie REC.

```
begin
| Comment : avant d'incrémenter la variable STKCRX, il faut sauver la
|           valeur de l'indice IDTCRX, dans la zone réservée au poin-
|           teur CRP. Si le pointeur RCP correspondant au noeud dont
|           on désire générer le contenu est égal à 0, cela signifie
|           que le contenu de ce noeud est vide.
|           Avant chaque appel récursif de la procédure TRTGEN, il
|           faut sauver la valeur des variables NTBCRX et ASKKPX dans
|           la pile de sauvetage SSK. Après l'exécution de chaque
|           procédure TRTGEN, on récupère les valeurs de IDTCRX,NTBCRX
|           et ASKKPX.
| - stk[stkcrx].crp := idtcrx ;
| - stkcrx := stkcrx + 1 ;
| - if ntb[ntbcrx].rcp ≠ 0
|       then begin
|            | - stk[stkcrx].nxr := ntb[ntbcrx].rcp ;
|            | - stk[stkcrx].fnm := ntb[ntb[ntbcrx].rtp].nnm ;
|            | - repeat
|            |     begin
|            |     | - RD-LIN(stk[stkcrx]) ;
|            |     | - while idtcrx ≠ idtmxl
|            |     |     do begin
|            |     |     | - GETCAR(stk[stkcrx]) ;
|            |     |     | - if crtcar ≠ '/'
|            |     |     |   then PUTCAR(crtadr)
|            |     |     |   else begin
|            |     |     |        | - GETCAR(stk[stkcrx]) ;
|            |     |     |        | - if crtcar = 'g'
|            |     |     |        |   then
|            |     |     |        |   begin
|            |     |     |        |   | - GETCAR(stk[stkcrx]) ;
|            |     |     |        |   | - ssk[sskcrx].ntbcrx := ntbcrx ;
|            |     |     |        |   | - ssk[sskcrx].askkpx := askkpx ;
|            |     |     |        |   | - TRTGEN(crtadr) ;
|            |     |     |        |   | - stkcrx := stkcrx - 1 ;
|            |     |     |        |   | - idtcrx := stk[stkcrx].crp ;
|            |     |     |        |   | - sskcrx := sskcrx - 1 ;
|            |     |     |        |   | - ntbcrx := ssk[sskcrx].ntbcrx ;
|            |     |     |        |   | - askkpx := ssk[sskcrx].askkpx ;
|            |     |     |        |   end
|            |     |     |        |   else erreur de syntaxe
|            |     |     |        |   fi
|            |     |     |        end
|            |     |     |   fi
|            |     |     end
|            |     |   od
|            |     end
|            |   until stk[stkcrx].nxr = 0
|            end
|   fi
end
```

3. Procedure TRTEND

```
Comment : cette procédure (treatment end) traite un ordre de
          fin de création d'un noeud.
          Elle utilise la procédure GETSGNCAR qui met à jour
          la position courante et fournit le premier carac-
          tère non blanc de l'enregistrement physique courant
          suivant la position courante.

begin
|
|Comment : syntaxe de la commande END  : / E {<nom de noeud>}0^1
|          La valeur de la variable CRTLEV est toujours égale
|          au numéro de niveau plus un du dernier noeud créé
|          dans une structure arborescente,en suivant le
|          parcours dynastique.
|          A défaut de < nom de noeud>, la commande END signale
|          qu'on vient de terminer la création d'une feuille
|          de l'arbre et qu'on s'attend à créer un nouveau
|          noeud comme étant le frère cadet du noeud créé
|          précédemment; il suffit donc de diminuer de 1 la
|          valeur de CRTLEV. Dans le cas contraire, on s'at-
|          tend à créer un nouveau noeud comme étant le frère
|          cadet du noeud spécifié par < nom de noeud > ; la
|          valeur de CRTLEV sera donc égale au numéro de ni-
|          veau de ce noeud.
|          Pour terminer la création d'une structure d'arbre
|          on a le choix entre deux possibilités : soit écri-
|          re plusieurs commandes / E successives, soit écri-
|          re une seule commande END en spécifiant le nom de
|          la racine de l'arbre. Pour éviter le danger d'écri-
|          re une commande END en trop, on a prévu le test
|          d'erreur dans la première instruction if.
|
|- if crtcar = 'eoln'
|     then begin
|          |- crtlev := crtlev - 1 ;
|          |- if crtlev < 0 then message d'erreur, erreur de
|          |                     fin de création d'une struc-
|          |                     ture arborescente.
|          |  fi
|          end
|     else begin
|          |Comment : on affecte la chaîne de caractères
|          |          <nom de noeud>à la variable CRN.
|          |- for cmncrx := 1 to nnmmxl
|          |      do crn[crncrx] := ' '  od ;
|          |- crncrx := 0 ;
|          |- while crtcar ≠ 'eoln'
|          |        do  begin
|          |            |- crncrx := crncrx + 1 ;
|          |            |- crn[crncrx] := crtcar ;
|          |            |- GETSGNCAR(stk[stkcrx])
|          |            end
|          |       od;
```

```
        Comment : on part du dernier noeud créé dans la
                  structure arborescente (LND) et on re-
                  monte dans la table des noeuds (NTB) à
                  l'aide des pointeurs père (FAP) pour
                  retrouver le noeud mentionné dans la
                  commande END. Si le numéro de niveau de
                  ce noeud est égal à 0, on a terminé la
                  création de la structure arbirescente.
        - ntbcrx := lnd ;
        - while (ntb[ntbcrx].nnm ≠ crn)
                and (ntb[ntbcrx].lnb ≠ 0)
                do  ntbcrx := ntb[ntbcrx].fap od ;
        - if (ntb[ntbcrx].lnb = 0)
             and (ntb[ntbcrx].nnm = crn)
          then begin
                 - crtlev := 0 ;
                 - envoyer un message de fin de création
                           de la structure arborescente
               end
          fi ;
        - if (ntb[ntbcrx].lnb ≠ 0)
             and (ntb[ntbcrx].nnm = crn)
          then crtlev := ntb[ntbcrx].lnb
          fi ;
        - if (ntb[ntbcrx].lnb = 0)
             and (ntb[ntbcrx].nnm ≠ crn)
          then message d'erreur,le noeud n'existe pas
          fi
      end
end
```

---

4. Procedure TRTINS

Comment : cette procédure (treatment insertion) traite un ordre d'insertion.
Elle utilise les procédures suivantes :
GETSGNCAR qui met à jour la position courante et fournit le premier caractère non blanc de l'enregistrement courant suivant la position courante.
GETCAR qui met à jour la position courante et fournit le caractère correspondant à cette position.
SETCRN qui affecte la chaîne de caractères ‹nom de noeud› à la variable CRN.

```
            TRTNID qui analyse un identifiant de noeud.
            TRT-EL qui traite l'insertion d'un nouveau noeud
                   comme fils ainé d'un noeud déjà existant.
            TRT-YO qui traite l'insertion d'un nouveau noeud
                   comme fils cadet d'un noeud déjà existant.
            TRT-YB qui traite l'insertion d'un nouveau noeud
                   comme frère cadet d'un noeud déjà existant.
            TRTCONNOD qui traite le contenu d'un noeud en
                      laissant intactes les commandes GENERATE
                      incorporées dans le contenu.
            TRTCONNODVAL qui traite le contenu d'un noeud par
                         valeur.

var sob : packed array[1..2] of char ;
    fstidx, lstidx, sobfpl : 0..ntbmxl ;
    fstlev : integer ;

begin
|Comment : syntaxe de la commande INSERT
|          / I <V> | <R> ( {!}^0_1 < identifiant de noeud > )
|              <lien de parenté>,<nom de noeud>{" contenu "}^1_0
|          <lien de parenté> ::= <fils ainé> | <fils cadet> |
|                                <frère cadet>
|          <fils ainé> ::=  EL
|          <fils cadet>::=  YO
|          <frère cadet> ::=  YB
|          La variable SOB (son or brother) indiquera le lien
|          de parenté.
|          La variable FSTIDX (first index) contiendra
|          l'adresse logique dans la table des noeuds du
|          noeud auquel on désire ajouter un fils ainé, un
|          fils cadet ou un frère cadet; la variable FSTLEV
|          (first level) indiquera le numéro de niveau de ce
|          noeud.
|          La variable LSTIDX (last index) contiendra l'adres-
|          se logique dans la table des noeuds du dernier noeu
|          du sous-arbre du noeud auquel on désire ajouter
|          un fils cadet ou un frère cadet.
|          Le contenu de la variable SOBFPL est identique à
|          celui du pointeur des places libres dans la table
|          des noeuds : NTBFRP.
|
|- rnb := 0 ;
|- if crtcar = 'v' then begin
|                       |- vrf-sw := true ;
|                       |- GETSGNCAR(stk[stkcrx])
|                       end
|                  else if crtcar = 'r'
|                       then begin
|                            |- vrf-sw := false ;
|                            |- GETSGNCAR(stk[stkcrx])
|                            end
|                       else erreur de syntaxe
|                       fi
|  fi
```

```
- if crtcar = '('
     then begin
          - GETSGNCAR(stk[stkcrx]) ;
          - TRTNID ;
          - if (mny = false) and (nnb > 1)
               then message d'erreur, < identifiant de noeud >
                    désigne une famille de noeuds
               else if nnb = 0
                    then message d'erreur, il n'y a aucun
                         noeud identifié par  identifiant
                         de noeud
                    fi
            fi ;
          - GETSGNCAR(stk[stkcrx])
          end
      else erreur de syntaxe
  fi ;
- if crtcar = 'y'
     then begin
          - GETSGNCAR(stk[stkcrx]) ;
          - if crtcar = 'b' then sob := 'yb' fi ;
          - if crtcar = 'o' then sob := 'yo'
                            else erreur de syntaxe
            fi
          end
     else if crtcar = 'e'
          then begin
               - GETSGNCAR(stk[stkcrx]) ;
               - if crtcar = 'l' then sob := 'el'
                                 else erreur de syntaxe
                 fi
               end
           else erreur de syntaxe
           fi
  fi ;
- GETSGNCAR(stk[stkcrx]) ;
- if crtcar = ',' then begin
                       - GETSGNCAR(stk[stkcrx]) ;
                       - SETCRN
                       end
                  else erreur de syntaxe
  fi ;
```

```
- repeat  begin
            - case sob of yb : TRT-YB ;
                          yo : TRT-YO ;
                          el : TRT-EL
              end ;
            - if vrf-sw = false then TRTCONNOD(sobfpl)
                               else TRTCONNODVAL(sobfpl)
              fi
          end
  until askcrx = askkpx
end
```

4.1. Procedure TRT-YB

```
Comment : cette procédure (treatment young brother) traite
          l'insertion d'un nouveau noeud comme frère cadet
          d'un noeud déjà existant.

begin
- askcrx := askcrx - 1 ;
- fstidx := ask[askcrx] ;
- sobfpl := ntbfrp ;
- fstlev := ntb[fstidx].lnb ;
Comment : pour mettre à jour le parcours dynastique dans la
          table des noeuds, il faut retrouver le dernier
          noeud (adresse = LSTIDX) du sous-arbre du noeud
          auquel on désire ajouter un frère cadet.
          A partir de l'adresse indiquée par la variable
          FSTIDX, on effectue le parcours dynastique jusqu'
          au moment où le numéro de niveau du noeud courant
          cesse d'être supérieur à la valeur de FSTLEV.
          Il reste alors à mettre à jour la table des noeuds
          et le pointeur NTBFRP.
- ntbcrx := ntb[fstidx].nxp ;
- while ntb[ntbcrx].lnb > fstlev
        do begin
             - lstidx := ntbcrx ;
             - ntbcrx := ntb[ntbcrx].nxp
           end
        od ;
- ntb[sobfpl].nnm := crn ;
- ntb[sobfpl].lnb := fstlev ;
- ntb[sobfpl].fap := ntb[fstidx].fap ;
- ntb[sobfpl].rtp := ntb[fstidx].rtp ;
- ntbfrp := ntb[sobfpl].nxp ;
- ntb[sobfpl].nxp := ntb[lstidx].nxp ;
- ntb[lstidx].nxp := sobfpl
end
```

4.2. Procedure TRT-YO

```
Comment : cette procédure (treatment youger) traite l'inser-
          tion d'un nouveau noeud comme fils cadet d'un noeud
          déjà existant.

begin
|- askcrx := askcrx - 1 ;
|- fstidx := ask[askcrx] ;
|- sobfpl := ntbfrp ;
|- fstlev := ntb[fstidx].lnb ;
|Comment : pour mettre à jour le parcours dynastique dans la
|          table des noeuds, il faut retrouver le dernier
|          noeud du sous-arbre du noeud auquel on désire
|          ajouter un fils cadet.
|- ntbcrx := ntb[ntbcrx].nxp ;
|- while ntb[ntbcrx].lnb > fstlev
|        do begin
|           |- lstidx := ntbcrx ;
|           |- ntbcrx := ntb[ntbcrx].nxp
|           end
|        od ;
|Comment : l'instruction conditionnelle if est nécessaire
|          pour la mise à jour du pointeur RTP correspondant
|          au nouveau noeud car nous avons pris la conven-
|          tion que le pointeur RTP correspondant à un nom
|          de racine contenait toujours l'adresse logique
|          dans la table des noeuds du dernier noeud créé
|          dans une structure arborescente.
|- ntb[sobfpl].nnm := crn ;
|- ntb[sobfpl].lnb := fstlev + 1 ;
|- ntb[sobfpl].fap := fstidx ;
|- if fstlev = 0 then ntb[sobfpl].rtp := fstidx
|                 else ntb[sobfpl].rtp := ntb[fstidx].rtp
|  fi ;
|- ntbfrp := ntb[sobfpl].nxp ;
|- ntb[sobfpl].nxp := ntb[lstidx].nxp ;
|- ntb[lstidx].nxp := sobfpl
end
```

4.3. Procedure TRT-EL

```
Comment : cette procédure (treatment elder) traite l'insertion
          d'un nouveau noeud comme fils ainé d'un noeud déjà
          existant.

begin
| - askcrx := askcrx - 1 ;
| - fstidx := ask[askcrx] ;
| - sobfpl := ntbfrp ;
| - fstlev := ntb[fstidx].lnb ;
| Comment : la mise à jour du parcours dynastique est immé-
|           diate puisqu'un fils ainé est toujours le suivant
|           de son père. On peut directement complèter la
|           table des noeuds.
| - ntb[sobfpl].nnm := crn ;
| - ntb[sobfpl].lnb := fstlev + 1 ;
| - ntb[sobfpl].fap := fstidx ;
| Comment : comme dans la procédure TRT-YO, l'instruction if
|           est nécessaire pour la mise à jour du pointeur RTP
|           correspondant au nouveau noeud.
| - if fstlev = 0 then ntb[sobfpl].rtp := fstidx
|                 else ntb[sobfpl].rtp := ntb[fstidx].rtp
|   fi ;
| - ntbfrp := ntb[sobfpl].nxp ;
| - ntb[sobfpl].nxp := ntb[fstidx].nxp ;
| - ntb[fstidx].nxp := sobfpl
end
```

---

5. Procedure TRTDEL

```
Comment : cette procédure (treatment delete) traite un
          un ordre de suppression.
          Elle utilise les procédures suivantes :
          GETSGNCAR qui met à jour la position courante et
                    fournit le premier caractère non blanc
                    de l'enregistrement courant suivant la
                    position courante.
          TRTNID qui traite un identifiant de noeud.
          DEL qui supprime tous les noeuds de la famille de
              noeuds désignés par un identifiant de noeud.
```

```
begin
|
|  Comment : syntaxe d'une commande DELETE
|                 / D ( { ! }_0^1 < identifiant de noeud > )
|            La variable booléenne MNY a la valeur false lors-
|            que le symbole ! est présent et la variable entiè-
|            re NNB indique le nombre de noeuds identifiés par
|            < identifiant de noeud > .
|  - if crtcar = '(' then begin
|                          |  - GETSGNCAR(stk[stkcrx]) ;
|                          |  - TRTNID
|                          end
|                     else erreur de syntaxe
|    fi ;
|  - if (mny = false) and (nnb > 1)
|       then message d'erreur, < identifiant de noeud > désigne
|            une famille de noeuds
|       else if nnb = 0
|            then message d'erreur, il n'y a aucun noeud iden-
|                 tifié par < identifiant de noeud >
|            else DEL
|            fi
|    fi
end
```

5.1. Procedure DEL

```
Comment : cette procedure (delete) supprime tous les noeuds
          d'un famille de noeuds désignés par < identifiant
          de noeud > .
          Elle utilise les procédures suivantes :
          DELALL qui supprime tous les noeuds d'un fichier à
                 structure arborescente.
          DELNDS qui supprime tous les noeuds du sous-arbre
                 de l'un des noeuds de la famille des noeuds
                 désignés par un < identifiant de noeud > .

var fstidx, lstidx : 0..ntbmxl ;
    fstlev : integer ;

Comment : la variable FSTIDX indique l'adresse logique dans
          la table des noeuds du noeud courant dont on dési-
          re supprimer le sous-arbre; la variable FSTLEV
          indique le numéro de niveau de ce noeud.
          La variable LSTIDX indique l'adresse logique dans
          la table des noeuds du dernier noeud(suivant le
          parcours dynastique) du sous-arbre qu'on désire
          supprimer. Cette variable sera utilisée pour mettre
          à jour le pointeur NTBFRP.
          La pile des adresses ASK contient les adresses
          logiques dans la table des noeuds de tous les
          noeuds de la famille de noeuds désignés par l'i-
          dentifiant de noeud.
```

```
                La suppression d'un noeud consiste à mettre à
                blanc l'élément correspondant dans la table des
                noeuds, à ajouter cet élément à la liste des pla-
                ces libres dans la table des noeuds et à libérer
                tous les enregistrements physiques du contenu de
                ce noeud dans la table partielle des places libres.

begin
|- repeat begin
|          |- askcrx := askcrx - 1 ;
|          |- fstidx := ask[askcrx] ;
|          |- fstlev := ntb[fstidx].lnb ;
|          |- if fstlev = 0 then DELALL else DELNDS fi
|          end
|  until askcrx = askkpx ;
|- stk[1].fnm := fsk[1]
|Comment : à la fin de l'exécution de cette procédure, on
|          doit récrire dans la zone FNM du premier élément
|          de la pile STK le nom du fichier d'entrée F0.
end
```

5.1.1. Procedure DELALL

```
Comment : cette procédure (delete all) supprime tous les
          noeuds d'un fichier à structure arborescente.

begin
|Comment : il faut retrouver la table partielle des places
|          libres de ce fichier et l'adresse courante dans
|          la table des racines (RTB) de l'élément corres-
|          pondant à ce nom de fichier.
|- rtbcrx := ntb[fstidx].fap ;
|- ftbcrx := rtb[rtbcrx].ftbptr ;
|Comment : la suppression d'un fichier entraîne la mise à
|          blanc de l'élément de la table des racines cor-
|          respondant à ce nom de fichier.
|- for rnmcrx := 1 to rnmmxl
|      do rtb[rtbcrx][rnmcrx]:= ' ' od ;
|Comment : il faut libérer , dans la table partielle des
|          places libres, tous les enregistrements physiques
|          de ce fichier à l'exception des BNB + 1 premiers
|          qui sont toujours réservés.
|- for fplcrx := bnb + 2 to fplmxl
|      do ftb[ftbcrx][fplcrx] := true od ;
|Comment : il faut retrouver le dernier noeud du parcours
|          dynastique de ce fichier pour rajouter tous les
|          éléments de la table partielle des noeuds de ce
|          fichier à la liste des places libres.
|- lnd := ntb[fstidx].rtp ;
```

```
|- ntb[lnd].nxp := ntbfrp ;
|- ntbfrp := fstidx ;
|Comment : il faut mettre à blanc toutes les zones NNM de la
|          table partielle des noeuds de ce fichier.
|- ntbcrx := fstidx ;
|- repeat begin
|          |- for nnmcrx := 1 to nnmmxl
|          |       do ntb[ntbcrx][nnmcrx] := ' ' od ;
|          |- ntbcrx := ntb[ntbcrx].nxp
|          end
|  until ntbcrx = 0
end
```

5.1.2. Procedure DELNDS

```
Comment : cette procédure (delete nodes) supprime tous les
          noeuds du sous-arbre de l'un des noeuds de la fa-
          mille de noeuds identifiés par un identifiant de
          noeud.
          Elle utilise la procédure RD-LIN qui lit un enre-
          gistrement physique courant à partir d'un fichier
          d'entrée.

begin
|Comment : il faut retrouver la table partielle des places
|          libres de ce fichier.
|- rtbcrx := ntb[ntb[fstidx].rtp].fap ;
|- ftbcrx := rtb[rtbcrx].ftbptr ;
|Comment : pour mettre à jour cette table partielle,il fau-
|          dra pour chaque noeud du sous-arbre, lire tous les
|          enregistrements physiques du contenu de ce noeud.
|          Le dernier enregistrement contient dans la zone
|          réservée au pointeur NXR la valeur 0.
|          Pour lire ces enregistrements, il faut au préala-
|          ble garnir la zone FNM du premier élément de la
|          pile STK avec le nom du fichier sur lequel on dé-
|          sire lire.
|- stk[1].fnm := ntb[ntb[fstidx].rtp].nnm ;
|- ntbcrx := fstidx ;
|- repeat begin
|          |- stk[1].nxr := ntb[ntbcrx].rcp ;
|          |Comment : mise à blanc de la zone NNM de l'élément
|          |          NTB[NTBCRX]de la table des noeuds.
|          |- for nnmcrx := 1 to nnmmxl
|          |       do ntb[ntbcrx][nnmcrx] := ' ' od ;
```

```
        Comment : avant de libérer les enregistrements
                  physiques d'un contenu de noeud, il faut
                  tester au préalable l'existence d'un
                  contenu.
        - if stk[1].nxr ≠ 0
             then repeat begin
                           - ftb[ftbcrx][stk[1].nxr] := true;
                           - RD-LIN(stk[1])
                         end
                  until stk[1].nxr = 0
          fi ;
        - lstidx := ntbcrx ;
        - ntbcrx := ntb[ntbcrx].nxp
      end
  until ntb[ntbcrx].lnb = fstlev ;
Comment : il reste à chaîner les éléments libérés dans la
          table des noeuds à la liste des places libres.
- ntb[lstidx].nxp := ntbfrp ;
- ntbfrp := ntb[fstidx].nxp
end
```

---

6. Procedure TRTADD

```
Comment : cette procédure (treatment ADD) traite un ordre de
          concaténation de contenus.
          Elle utilise les procédures suivantes :
          GETSGNCAR qui met à jour la position courante et
                    fournit le premier caractère non blanc
                    de l'enregistrement physique courant
                    suivant la position courante.
          TRTNID qui traite un identifiant de noeud.
          ADDCON qui complète le contenu de chacundes noeuds
                 de la famille de noeuds identifiés par un
                 identifiant de noeud par le texte du conte-
                 nu mentionné dans la commande ADD.

begin
Comment : syntaxe d'un commande ADD
              / A ( {!}_0^^ < identifiant de noeud > ) " < contenu >"
          La variable booléenne MNY a la valeur false lors-
          que le symbole ! est présent. La variable entière
          NNB indique le nombre de noeuds identifiés par un
          identifiant de noeud
```

```
- if crtcar = '(' then begin
                        - GETSGNCAR(stk[stkcrx]) ;
                        - TRTNID ;
                        - GETSGNCAR(stk[stkcrx])
                       end
  fi ;
- if (mny = false) and (nnb > 1)
     then message d'erreur, < identifiant de noeud > désigne
          une famille de noeuds
     else if nnb = 0
          then message d'erreur, il n'y a aucun noeud iden-
               tifié par < identifiant de noeud >
          else ADDCON
          fi
  fi
end
```

6.1. Procedure ADDCON

```
Comment : cette procédure complète le contenu de chacun des
          noeuds de la famille des noeuds désignés par un
          identifiant de noeud par le texte du contenu mention-
          né dans la commande ADD.
          Elle utilise les procédures suivantes :
          RD-LIN qui lit l'enregistrement courant à partir
                 du fichier d'entrée.
          TRTCONNOD qui traite le contenu d'un noeud en lais-
                    sant intactes les commandes GENERATE incor-
                    porées dans le texte du contenu.
var fstidx : 0..ntbmxl
    fstlev : integer
Comment : la pile des adresse ASK contient l'adresse logique
          dans la table des noeuds de chaque noeud de la fa-
          mille de noeuds désignés par un identifiant de noeud.
          La variable FSTIDX indique l'adresse logique dans
          la table de noeuds du noeud courant auquel on désire
          appliquer un ordre de concaténation, la variable
          FSTLEV indique le numéro de niveau de ce noeud.
begin
- repeat begin
          - askcrx := askcrx - 1 ;
          - fstidx := ask [askcrx] ;
          - fstlev := ntb [fstidx] .lnb ;
          Comment : la zone FNM du premier élément de la pile
                    STK contient toujours le nom du fichier
                    dans lequel on va lire des enregistrements
                    Si le noeud, auquel on désire ajouter
```

```
                    un nouveau contenu possède déjà un con-
                    tenu, il faut aller rechercher le dernier
                    enregistrement physique du contenu de ce
                    noeud avant d'effectuer la concaténation.
                    Pour effectuer cette séquence de lecture,
                    il faut au préalable mettre à jour la
                    zone FNM du premier élément de la pile
                    STK avec le nom du fichier dans lequel
                    on va lire.
                    Si le noeud ne possède pas de contenu,
                    on peut directement appeler la procédure
                    TRTCONNOD puisque la zone FNM contient
                    déjà le nom du fichier d'entrée FO.
          - if ntb[fstidx].rcp = 0
            then TRTCONNOD (fstidx)
            else begin
                 | - if fstlev = 0
                   | then stk[1].fnm := ntb[fstidx].nnm
                   | else stk[1].fnm := ntb[ntb[fstidx].rtp].n
                   | fi ;
                 | - stk[1].nxr := ntb[fstidx].rcp ;
                 | - rnb := 0 ;
                 | - repeat begin
                   |          | - RD-LIN(stk[1]) ;
                   |          | - rnb := rnb + 1
                   |          end
                   | until stk[1].nxr = 0 ;
                 | Comment : il faut retrouver le caractère
                   |         etx du dernier enregistrement
                   |         avant de pouvoir effectuer la
                   |         concaténation ; la zone FNM du
                   |         premier élément de la pile STK
                   |         doit contenir le nom du fichier
                   |         d'entrée FO.
                 | - idtcrx := 1 ;
                 | - while stk[1].idt[idtcrx] ≠ etx
                   |       do idtcrx := idtcrx + 1  od ;
                 | - idtcrx := idtcrx - 2 ;
                 | - stk[1].fnm := fsk[1];
                 | - TRTCONNOD(fstidx)
                 end
              fi
        end
  until askcrx = askkpx
end
```

7. Procedure TRTMOD

```
Comment : cette procédure (treatment modify) traite un ordre
          de modification.
          Elle utilise les procédures suivantes :
          GETSGNCAR     qui met à jour la position courante
                    et fournit le premier caractère non blanc
                    de l'enregistrement physique courant sui-
                    vant la position courante.
          TRTNID qui traite un identifiant de noeud
          SETCRN qui affecte la chaîne de caractères < nom de
                 noeud > à la variable CRN.
          MODNNM qui modifie le nom d'un noeud.
          MODCON qui modifie le contenu d'un noeud.
var fstidx : 0..ntbmxl ;
    prsnnm, prscon : boolean ;
Comment : la variable FSTIDX contient l'adresse logique dans
          la table des noeuds du noeud courant dont on dési-
          re modifier le nom et/ou le contenu.
          La variable booléenne PRSNNM a la valeur true
          lorsqu'on désire modifier le nom d'un noeud et la
          variable PRSCON a la valeur true lorsqu'on désire
          modifier le contenu d'un noeud.
begin
|Comment : syntaxe d'une commande MODIFY
|          / M ( {!}0^ < identifiant de noeud > ) < modification >
|          < modification > ::= < nom de noeud > | "<contenu>" |
|                                < nom de noeud > "<contenu>"
|          La variable booléenne MNY a la valeur false
|          lorsque le symbole ! est présent et la variable
|          entière NNB indique le nombre de noeuds identi-
|          fiés par  identifiant de noeud .
|- if crtcar = '(' then begin
|                        |- GETSGNCAR(stk[stkcrx]) ;
|                        |- TRTNID ;
|                        |- GETSGNCAR(stk[stkcrx])
|                        end
|  fi ;
|- if (mny = false) and (nnb > 1)
|     then message d'erreur, < identifiant de noeud >désigne
|          une famille de noeuds
|     else if nnb = 0
|          then message d'erreur, il n'y a aucun noeud iden-
|               difié par  identifiant de noeud
|          fi
|   fi ;
|- if crtcar = '"' then prsnnm := false
|                  else begin
|                       |- prsnnm := true ;
|                       |- SETCRN
|                       end
|  fi ;
```

```
- if crtcar = 'eoln' then prscon := false
                     else prscon := true
  fi ;
- repeat begin
          - askcrx := askcrx - 1 ;
          - fstidx := ask[askcrx] ;
          - if prsnnm = true then MODNNM fi ;
          - if prscon = true then MODCON fi
          end
  until askcrx = askkpx
end
```

7.1. Procedure MODNNM

```
Comment : cette procédure (modify node name) modifie le nom
          d'un noeud.

begin
- crncrx := 1 ;
- for nnmcrx := 1 to nnmmx1
      do begin
          - ntb[fstidx].nnm[nnmcrx] := crn[crncrx];
          - crncrx := crncrx + 1
          end
      od
end
```

7.2. Procedure MODCON

```
Comment : cette procedure(modify content) modifie le contenu
          d'un noeud.
          Elle utilise les procédures suivantes :
          RD-LIN qui lit un enregistrement physique courant
                 à partir du fichier d'entrée.
          TRTCONNOD qui traite le contenu d'un noeud en
                    laissant intactes les commandes GENERATE
                    incorporées dans le texte du contenu.

begin
Comment : pour modifier le contenu d'un noeud, on va d'abord
          libérer dans la table des places libres tous les
          emplacements réservés aux enregistrements physiques
          du contenu de ce noeud.
```

```
        Pour mettre à jour la table partielle des places
        libres, il faut lire tous les enregistrements
        physiques du contenu de ce noeud. Le dernier en-
        registrement contient dans la zone réservée au
        pointeur NXR la valeur 0.
        Pour lire ces enregistrements, il faut au préalable
        garnir la zone FNM du premier élément de la pile
        SSK avec le nom du fichier sur lequel on désire
        lire. Si le noeud ne possède pas de contenu, on
        peut directement appeler la procedure TRTCONNOD.
- if ntb[fstidx].rcp = 0
     then TRTCONNOD(fstidx)
     else begin
          Comment : on recherche la table partielle des
                    places libres.
          - rtbcrx := ntb[ntb[fstidx].rtp].fap ;
          - ftbcrx := rtb[rtbcrx].ftbptr ;
          - stk[1].fnm := ntb[ntb[fstidx].rtp].nnm;
          - stk[1].nxr := ntb[fstidx].rcp ;
          - repeat begin
                    - ftb[ftbcrx][stk[1].nxr] := true ;
                    - RD-LIN(stk[1])
                    end
            until stk[1].nxr = 0 ;
          - rnb := 0 ;
          - stk[1].fnm := fsk[1] ;
          - TRTCONNOD(fstidx)
          end
   fi
end
```

PARAGRAPHE D : TERMINAISON

Procedure TRTTER

```
Comment : cette procédure (treatment termination) recharge
          en mémoire secondaire les tables partielles des
          places libres et les tables partielles des noeuds
          de tous les fichiers à structure arborescente
          traités dans notre application.
          Comme les éléments de chaque table partielle des
          noeuds seront enregistrés suivant l'ordre du par-
          cours dynastique, il faudra recopier en mémoire
          centrale la table partielle des noeuds de chaque
          fichier afin de mettre à jour le pointeur FAP de
          chaque noeud.
          Elle utilise les procédures suivantes :
          UPDFAP qui recopie en mémoire centrale la table
                 partielle des noeuds d'un fichier en mettant
                 à jour les pointeurs FAP.
          RCHNTB qui procède à l'enregistrement en mémoire
                 secondaire de la table partielle des noeuds
                 d'un fichier.
          RCHFTB qui procède à l'enregistrement en mémoire
                 secondaire de la table partielle des places
                 libres d'un fichier.

const pntmxl ;

type pntent = record  nnm : packed array [1..nnmmxl] of char
                      lnb : integer ;
                      rtp : o..ntbmxl ;
                      nxp : o..ntbmxl ;
                      fap : o..ntbmxl ;
                      rcp : integer
              end ;

var pnt : array [1..pntmxl] of pntent ;
    crtfnm : packed array [1..nnmmxl] of char ;
    adr : integer ;
    pntcrx, pntkpx : o..pntmxl ;
    crtlev : integer ;

Comment : le tableau PNT (partiel note table) contiendra la
          table partielle des noeuds d'un fichier avec les
          pointeurs FAP mis à jour.
          La constante PNTMXL désigne la taille maximum de
          ce tableau.
          La variable PNTKPX sert à mémoriser la valeur de
          l'indice PNTCRX du tableau PNT.
```

```
begin
|Comment : on parcourt séquentiellement toute la table des
|          racines et on utilise les pointeurs NTBPTR, FTBPTR
|          pour retrouver la table partielle des noeuds et
|          la table partielle des places libres de chaque
|          fichier.
|- rtbcrx := 1 ;
|- while rtbcrx ≤ rtbmxl
|        do begin
|           |- if rtb [rtbcrx] .rnm = ' '
|           |  then rtbcrx := rtbcrx + 1
|           |  fi ;
|           |- ntbcrx := rtb [rtbcrx ] .ntbptr ;
|           |- ftbcrx := rtb [rtbcrx ] .ftbptr ;
|           |- crtfnm := ntb [ntbcrx ] .fnm ;
|           |- UPDFAP
|           |- RCHNTB
|           |- RCHFTB
|           end
|        od
end
```

1. Procedure UPDFAP

```
Comment : cette procédure (update father pointer) recopie
          en mémoire centrale la table partielle des noeuds
          d'un fichier en mettant à jour les pointeurs FAP.
var pntidx : 0..pntmxl ;
begin
|- pntcrx := 0 ;
|- repeat begin
|          |- pntkpx := pntcrx ;
|          |- pntcrx := pntcrx + 1 ;
|          |Comment : le pointeur RTP correspondant à un nom
|          |          de racine sera mis à jour à la fin de
|          |          la procédure UPDFAP, puisqu'il doit
|          |          contenir l'adresse logique dans la table
|          |          partielle des noeuds,du dernier noeud
|          |          du parcours dynastique du fichier.
|          |
|          |
|          |
```

```
        - if ntb[ntbcrx].lnb = 0
            then begin
                   - pnt[pntcrx].nnm := ntb[ntbcrx].nnm ;
                   - pnt[pntcrx].lnb := ntb[ntbcrx].lnb ;
                   - pnt[pntcrx].nxp := pntcrx + 1 ;
                   - pnt[pntcrx].rcp := ntb[ntbcrx].rcp ;
                 end
            else begin
                   - crtlev := pnt[pntkpx].lnb ;
                   - pnt[pntcrx].nnm := ntb[ntbcrx].nnm ;
                   - pnt[pntcrx].lnb := ntb[ntbcrx].lnb ;
                   - pnt[pntcrx].rtp := 1 ;
                   - pnt[pntcrx].nxp := pntcrx + 1 ;
                   - pnt[pntcrx].rcp := ntb[ntbcrx].rcp ;
                   Comment : pour mettre à jour le pointeur FAP de
                             l'élément courant(PNTCRX) de la table
                             PNT, on utilise la variable CRTLEV qui
                             indique le numéro de niveau de l'élé-
                             ment précédent (PNTKPX = PNTCRX - 1) de
                             la table PNT.
                   - if pnt[pntcrx].lnb = crtlev
                     then pnt[pntcrx].fap := pnt[pntkpx].fap fi ;
                   - if pnt[pntcrx].lnb = crtlev + 1
                     then pnt[pntcrx].fap := pntkpx fi ;
                   Comment : si le numéro de niveau de l'élément
                             courant de la table PNT est inférieur
                             à la valeur de CRTLEV, on remonte dans
                             la table PNT à l'aide des pointeurs
                             FAP pour retrouver le noeud ayant le
                             même numéro de niveau que celui de
                             l'élément courant.
                   - if pnt[pntcrx].lnb<crtlev
                     then begin
                            - pntidx := pntkpx ;
                            - while pnt[pntidx].lnb > crtlev
                                do pntidx := pnt[pntidx].fap  od ;
                            - pnt[pntcrx].fap := pntidx
                          end
                     fi
                 end
          fi ;
        - ntbcrx := ntb[ntbcrx].nxp
      end
  until ntbcrx = 0 ;
Comment : il reste à mettre à jour le pointeur FAP  du premier
          élément de la table PNT
- pnt[1].fap := pntcrx
end
```

2. Procedure RCHNTB (var crtadr : integer)

```
Comment : cette procédure (recharge node table) enregistre
          en mémoire secondaire la table partielle des noeuds
          d'un fichier.
          Elle utilise les procédures suivantes :
          FNDFPL qui recherche ene place libre dans le fi-
                 chier de sortie
          SETREC qui garnit l'enregistrement de sortie REC.
          WRTNTB qui écrit l'enregistrement courant dans le
                 fichier de sortie.

begin
| Comment : l'adresse logique dans le fichier de sortie de
|           l'enregistrement physique qui contient le début de
|           la table partielle des noeuds est toujours égale
|           à BNB + 1 ; la constante BNB indique le nombre
|           d'enregistrements physiques du fichier réservé à
|           la table partielle des places libres.
|           La variable PNTCRX est mise à jour par la procédure
|           SETREC.
| - pntcrx := 1 ;
| - crtnxr := bnb + 1 ;
| - repeat begin
|           | - SETREC ;
|           | - if pntcrx = 0 then begin
|           |                      | - rec.nxr := 0 ;
|           |                      | - WRTNTB(crtnxr)
|           |                      end
|           |                 else begin
|           |                      | - FNDFPL(rec.nxr) ;
|           |                      | - WRTNTB(crtnxr)
|           |                      end
|           | - crtnxr := rec.nxr
|           end
|   until pntcrx = 0
end
```

2.1. Procedure SETREC

```
Comment : cette procédure (set record) garnit l'enregistre-
          ment de sortie REC.

var i, j, k  : integer ;
```

```
begin
|- k := 1 ; i := 1 ; j := nnmmx1 ;
|Comment : la constante NSE indique le nombre d'éléments de la
|          table des noeuds que peut contenir un enregistre-
|          ment physique.
|- while k ≠ nse + 1
|        do begin
|           |- nnmcrx := 1 ;
|           |- for odtcrx := i to j
|           |      do begin
|           |         |- odt[odtcrx ] := pnt[pntcrx].nnm[nnmcrx]
|           |         |- nnmcrx := nnmcrx + 1
|           |         end
|           |      od ;
|           |- i := j + 1 ; j := j + 3 ;
|           |- for odtcrx := i to j
|           |      do odt[odtcrx ] := pnt[pntcrx].lnb  od ;
|           |- i := j + 1 ; j := j + 3 ;
|           |- for odtcrx := i to j
|           |      do odt[odtcrx ] := pnt[pntcrx].rtp  od ;
|           |- i := j + 1 ; j := j + 3 ;
|           |- for odtcrx := i to j
|           |      do odt[odtcrx ] := pnt[pntcrx].nxp  od ;
|           |- i := j + 1 ; j := j + 3 ;
|           |- for odtcrx := i to j
|           |      do odt[odtcrx ] := pnt[pntcrx].fap  od ;
|           |- i := j + 1 ; j := j + 3 ;
|           |- for odtcrx := i to j
|           |      do odt[odtcrx ] := pnt[pntcrx].rcp  od ;
|           |- pntcrx := pnt[pntcrx].nxp ;
|           |- if pntcrx ≠ 0 then k := k + 1
|           |                else k := nse + 1
|           |  fi
|           end
end
```

2.2. Procedure FNDFPL (var crtadr : integer)

```
Comment : cette procédure (find free place) recherche une
          place libre dans le fichier de sortie.

begin
|- fplcrx := 1 ;
|- while (ftb[ftbcrx][fplcrx] = false) and (fplcrx ≤ fplmxl)
          do fplcrx := fplcrx + 1 od ;
|- if fplcrx > fplmxl then message d'erreur, plus de places
                            libres
|  fi ;
|- ftb[ftbcrx][fplcrx]:= false ;
|- crtadr := fplcrx
end
```

2.3. Procedure WRTNTB (var crtadr : integer)

```
Comment : cette procédure (write node table) écrit dans le
          fichier de sortie , l'enregistrement courant qui
          contient des éléments de la table partielle des
          noeuds.
          Elle utilise la procédure d'écriture WRITE.

begin
|- case crtfnm of fsk[2] : WRITE(F1,crtadr,rec) ;
|                  fsk[3] : WRITE(F2,crtadr,rec) ;
|                  fsk[4] : WRITE(F3,crtadr,rec) ;
|                  fsk[5] : WRITE(F4,crtadr,rec)
|   end
end
```

3. Procedure RCHFTB

```
Comment : cette procédure (recharge free table) enregistre
          en mémoire secondaire la table partielle des pla-
          ces libres d'un fichier.
          Elle utilise la procédure d'écriture WRITE.

var i : integer ;

Comment : dans chaque fichier, les BNB premiers enregistre-
          ments contiennent toujours la table partielle des
          places libres du fichier.
```

```
begin
|- fplcrx := 1 ;
|- i := 1 ;
|- repeat begin
|          |- for bitcrx := 1 to bitmx1
|          |        do begin
|          |             |- bit[bitcrx] := ftb[ftbcrx][fplcrx] ;
|          |             |- fplcrx := fplcrx + 1
|          |           end
|          |        od ;
|          |- case crtfnm of fsk[2] : WRITE(F1,i,bit) ;
|          |                 fsk[3] : WRITE(F2,i,bit) ;
|          |                 fsk[4] : WRITE(F3,i,bit) ;
|          |                 fsk[5] : WRITE(F4,i,bit)
|          |   end
|          |- i := i + 1
|          end
|   until i > bnb
end
```

CONCLUSION

CONCLUSION

Nous allons, au terme de ce mémoire, tirer certaines conclusions notamment en ce qui concerne les points suivants:

- critique de l'analyse organique
- extensions du travail
- ce qu'il reste à faire

1. Critique de l'analyse organique

Pour décrire les algorithmes des différentes procédures, nous avons voulu utiliser un pseudo-langage très proche du langage Pascal, espérant ainsi faciliter la tâche du programmeur. Cette optique, pour justifiée qu'elle est, nous paraît devoir être corrigée car les résultats auxquels nous avons abouti au terme de cette analyse nous montrent qu'il aurait été préférable d'adopter un langage s'écartant davantage du langage Pascal.

Citons quelques exemples:

- dans la procédure TRTITE (page 56), on peut supprimer l'instruction:

  ```
  If (crtcar ≠ 'c') or (crtcar ≠ 'e') or (crtcar ≠ 'i') or
  (crtcar ≠ 'd') or (crtcar ≠ 'a') or (crtcar ≠ 'm')
  then erreur de synthaxe
  fi
  ```

  si l'on admet d'ajouter à la série des choix multiples de l'instruction case un choix "any" suivi de l'instruction erreur de synthaxe.

- on peut tout aussi bien supprimer la pile FSK des noms de fichie (page 49) en offrant au pseudo-langage la possibilité de manipuler des variables "nom de fichier".

- pour mettre à blanc une variable déclarée 'packed array of char' NNM par exemple, il est plus simple et plus clair d'écrire

  ```
  NTB.NNM := ' ' en lieu et place de l'instruction :
  for NNMCRX := 1 to NNMMXL do NTB[NTBCRX][NNMCRX] := ' ' od.
  ```

2. Extensions possibles

Au stade actuel de notre travail, il nous est imposé d'écrire complètement le fichier des commandes FO avant de pouvoir appeler notre programme d'application. Il serait intéressant de rendre notre système interactif et de pouvoir le combiner avec un éditeur ce qui nous permettrait de travailler directement au niveau d'une commande.

3. Ce qu'il reste à faire

Outre l'implémentation qui ne faisait pas partie de ce travail, il reste deux points importants à résoudre:

- l'analyse des entrées-sorties physiques

- la gestion des erreurs

Tout au long de l'analyse organique, nous nous sommes contentés de signaler l'erreur à l'exploitant par un message; aucune récupération d'erreur n'est prévue.

ANNEXES

ABS : absence
ACT : action table
ADD : add
ADR : address
ALL : all
ANA : analyse
ARG : argument
ASK : stack of address
BIT : bit
BNB : number of array BIT
CAR : character
CHG : charge
CMN : common
CMP : compatible
COD : code
CON : content
CRE : create
CRF : current file
CRN : current name
CRP : current position
CRT : current
CRX : current index
DEL : delete
DGT : digit
DST : distance
END : end
ENT : entry
ERR : error
FAP : father pointer
FND : find
FNM : file name
FPL : free place
FRP : pointer to free
FSK : stack of file name
FST : first
FTB : free table
GEN : generate
GET : get

IDN : identical
IDT : input data
IDX : index
INS : insert
INT : initialisation
ITE : iteration
KPX : keep index
LET : letter
LEV : level
LIN : line
LNB : level number
LND : last node
LSK : stack of level number
LST : last
MNY : many
MOD : modify
MXL : maxlength
NBR : number
NDS : nodes
NID : identifier of node
NME : node name
NOD : node
NSE : number of STKENT
NSK : stack of node name
NTB : node table
NXP : pointer to next
NXR : next record
ODT : output data
OLD : old
ONE : one
PNT : partiel node table
PRS : present
PTR : pointer
PUT : put
RCH : recharge
RCP : record pointer
RD- : read
REC : record
RNB : record number
RNM : root name
ROT : root

RTB : root table
RTP : root pointer
SCH : search
SEC : second
SGN : significant
SOB : son or brother
SOL : solution
SSK : salvage stack
STA : star
STK : stack
STR : string
TBS : tables
TER : termination
TRS : transition table
TRT : treatment
TST : test
UPD : update
VAL : value
VRF : value or reference
WRT : write
-EL : elder
-SW : switch
-YB : young brother
-YO : younger

ANNEXE B : DICTIONNAIRE DES DONNEES.

Pour la description d'une donnée composée, on renverra à la liste de ses composants décrits par ailleurs.

ACTARG (argument action table) : array [0..4,1..5] of procedure
tableau de procédures associé au tableau TRSARG.

ACTNID (node identifier action table) : array [0..4,1..6] of procedure
tableau de procedures associé au tableau TRSNID.

ASK (stack of address) : array [1..ASKMXL] of integer
cette pile contiendra les adresses logiques dans la table des noeuds de tous les noeuds de la famille de noeuds désignés par un identifiant de noeud.

ASKCRX (stack of address current index) : 0..ASKMXL
indice de la pile ASK.

ASKKPX (stack of address keep index) : 0..ASKMXL
donnée de manoeuvre qui sert à mémoriser la hauteur dans la pile ASK du premier noeud de la famille de noeuds désignés par un identifiant.

ASKMXL (maxlength of ASK) : constante
taille maximum de la pile ASK.

BNB (bit number) : constante
nombre maximum d'enregistrements physiques réservés au début de chaque fichier pour contenir la table partielle des places libres de ce fichier.

BIT (bit) : packed array [1..BITMXL] of boolean
buffer de bits utilisé pour la lecture ou l'écriture des enregistrements physiques réservés aux tables partielles des places libres.

BITCRX (bit current index) : 0..BITMXL
indice du tableau BIT.

BITMXL (maxlength of BIT) : constante
taille maximum du tableau BIT.

BUF (buffer) : record (IDT,NXR)
composant de la donnée composée STKENT, buffer de lecture.

CRN (current name) : packed array [1..NNMMXL] of char
donnée de manoeuvre contenant le nom du noeud courant dans la table des noeuds.

CRNCRX (current name current index) : 0..NNMMXL
indice du tableau CRN.

CRP (current position) : 0..IDTMXL
composant de la donnée composée STKENT
donnée de manoeuvre qui mémorise la
valeur de l'indice IDTCRX du tableau IDT

CRTLEV (current level) : integer
donnée de manoeuvre dont le contenu est
égal au numéro de niveau plus 1 du der-
nier noeud crée dans une structure arbo-
rescente, en suivant le parcours dynas-
tique

CRTCAR (current character) : char
donnée de manoeuvre contenant le
caractère courant.

CRTNXR (current next record) : integer
donnée de manoeuvre qui mémorise la valeur
du pointeur NXR de REC

FAP (father pointer) : 0..NTBMXL
composant de la donnée composée NTBENT
pointeur contenant l'adresse logique dans
la table des noeuds du père d'un noeud.

FNM (file name) : packed array [1..NNMMXL] of char
composant de la donnée composée STKENT
donnée de manoeuvre qui permettra au système
de déterminer le fichier dans lequel on désire
lire.

FNMCRX (file name current index) : 0.. NNMMXL
indice du tableau FNM.

FPL (free place) : packed array [1..FPLMXL] of boolean
composant de la donnée composée FTB
tableau de bits contenant l'ensemble des BNB
premiers enregistrements physiques d'un fichi
à structure arborescente (mapping des places
libres dans le fichier).

FPLCRX (free place current index) : 0..FPLMXL
indice du tableau FPL.

FPLKPX (free place keep index) : 0..FPLMXL
donnée de manoeuvre qui mémorise la valeur
de l'indice FPLCRX.

FPLMXL (maxlength of FPL) : constante
taille maximum du tableau FPL.

FSK (stack of file name) : array [1..FSKMXL] of FSKENT
pile des noms de fichiers utilisés dans
notre application. Ces noms sont considérés
comme des chaînes de caractères; le langage
PASCAL n'offrant pas la possibilité de
manipuler des variables "nom de fichier".

FSKENT (entry of FSK) : packed array [1..NNMMXL] of char
élément de la pile FSK.

FSKCRX (FSK current index) : 0..FSKMXL
indice de la pile FSK.

FSKMXL (maxlength of FSK) : constante
taille maximum de la pile FSK. Elle est égale au nombre de fichiers utilisés dans notre application.

FTB (free table) : array [1..FTBMXL] of FPL
table des places libres constituée séquentiellement en mémoire centrale par l'ensemble des tables partielles des places libres de tous les fichiers traités dans notre application.

FTBCRX (free table current index) : 0..FTBMXL
indice du tableau FTB.

FTBKPX (free table keep index) : 0..FTBMXL
donnée de manoeuvre qui mémorise la valeur de l'indice FTBCRX.

FTBMXL (maxlength of FTB) : constante
taille maximum de la table des places libres. Elle est égale au nombre de fichiers utilisés dans notre application.

FTBPTR (pointer to FTB) : 0..FTBMXL
composant de la donnée composée RTBENT pointeur indiquant l'adresse logique dans la table des places libres (FTB) de la table partielle des places libres d'un fichier.

IDT (input data) : packed array [1..IDTMXL] of char
composant de la donnée composée BUF.

IDTCRX (input data current index) : 0..IDTMXL
indice du tableau IDT.

IDTMXL (maxlength of IDT) : constante
taille maximum du tableau IDT.

LNB (level number) : integer
composant de la donnée composée NTBENT. Elle indique le numéro de niveau du noeud courant.

LND (last node) : 0..NTBMXL
donnée de manoeuvre contenant l'adresse logique dans la table des noeuds du dernier noeud crée dans un fichier à structure arborescente en suivant le parcours dynastique.

MNY (many) : boolean
variable booléenne qui prendra la valeur true lorsqu'un identifiant de noeud désignera une famille de noeuds.

NNB (number of nodes) : integer
donnée de manoeuvre qui indique le nombre de noeuds identifiés par un identifiant de noeud.

NNM (node name) : packed array [1..NNMMXL] of char
composant de la donnée composée NTBENT. Elle indique le nom du noeud courant.

NNMCRX (node name current index) : 0..NNMMXL
indice du tableau NNM.

NNMMXL (maxlength of NNM) : constante
taille maximum d'un nom de noeud

NSE (number of stack entry) : constante
nombre maximum d'éléments STKENT que peut contenir un enregistrement physique.

NTB (node table) : array [1..NTBMXL] of NTBENT
table des noeuds constituée séquentiellement en mémoire centrale par l'ensemble des tables partielles des noeuds de tous les fichiers de notre application.

NTBCRX (node table current index) : 0..NTBMXL
indice de la table des noeuds NTB.

NTBENT (entry of node table) : record (NNM, LNB, RTP, NXP, FAP, RCP)
élément de la table des noeuds NTB

NTBFRP (node table free pointer) : 0..NTBMXL
pointeur qui indique la première place libre disponible dans la table des noeuds NTB.

NTBKPX (node table keep index) : 0..NTBMXL
donnée de manoeuvre qui mémorise la valeur de l'indice NTBCRX.

NTBMXL (maxlength of NTB) : constante
taille maximum de la table des noeuds NTB.

NTBPTR (pointer to node table) : 0..NTBMXL
composant de la donnée composée RTBENT
pointeur qui indique l'adresse logique dans la table des noeuds du début de la table partielle des noeuds d'un fichier

NXP (pointer to next) : 0..NTBMXL
composant de NTBENT
pointeur vers le noeud suivant dans le parcours dynastique.

NXR (next record) : integer
composant des données composées BUF et REC
pointeur vers l'enregistrement suivant du contenu d'un noeud.

```
ODT (output data) : packed array [1..ODTMXL] of char
                    composant de la donnée composée REC.

ODTCRX (output data current : 0..ODTMXL
        index)                 indice du tableau ODT.

ODTMXL (maxlength  of ODT) : constante
                             taille maximum du tableau ODT.

REC (record) : record (ODT, NXR)
               buffer d'écriture.

RECMXL (maxlength  of REC) : constante
                             taille maximum de REC.

RCP (record pointer) : integer
                       composant de la donnée composée NTBENT
                       pointeur vers le premier enregistrement
                       physique du contenu d'un noeud.

RNB (record number) : integer
                      donnée de manoeuvre qui sert à compter
                      le nombre d'enregistrements physiques du
                      contenu d'un noeud.

RNM (root name) : packed array [1..NNMMXL] of char
                  composant de la donnée composée RTBENT
                  donnée de manoeuvre contenant le nom de la racine
                  d'une structure arborescente.

RNMCRX (root name current : 0..RNMMXL
        index)              indice du tableau rnm.

RTB (root table) : array [1..RTBMXL] of RTBENT
                   table des racines.

RTBCRX (root table current : 0..RTBMXL
        index)               indice du tableau RTB.

RTBENT (root table : record (RNM, NTBPTR, FTBPTR)
        entry)       élément du tableau RTB.

RTBKPX (root table keep : 0..RTBMXL
        index)            donnée de manoeuvre qui mémorise la
                          valeur de l'indice RTBCRX.

RTBMXL (maxlength  of root : constante
        table)               taille maximum de la table des racines
                             (RTB).

RTP (root pointer) : 0..NTBMXL
                     composant de la donnée composée NTBENT
                     pointeur contenant l'adresse logique dans la
                     table des noeuds de la racine d'un noeud.

STK (stack) : array [1..STKMXL] of STKENT
              stack utilisé pour la récursivité des commandes
              generate.
```

STKCRX (stack current index) : 0..STKMXL
indice de STK.

STKENT (stack entry) : record (FNM, CRP, BVF)
élément de STK.

STKMXL (maxlength of STK) : constante
taille maximum du stack.

TRSARE (argument transition table) : array [0..4, 1..5] of integer
tableau de transition correspondant à la partie <argument> d'une command generate.

TRSNID (node identifier transition table) : array [0..4, 1..6] of integer
tableau de transition correspondant à un identifiant de noeud .

VRF-SW (value or reference switch) : boolean
variable booléenne qui prendra la valeur true lorsque l'élément <fonct de la synthaxe des commandes Create et Insert aura la valeur 'V'.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

1. A. CLARINVAL

   "Méthodologie de l'analyse"

   Institut d'Informatique Namur

2. C. CHERTON

   "Organisation des fichiers"

   Institut d'Informatique Namur

3. A. CLARINVAL

   "Cours de Cobol"

   Institut d'Informatique Namur

4. H. LEROY

   "Théorie des langages"

   Institut d'Informatique Namur

5. K. JENSON AND N. WIRTH

   "Pascal user manual and report"